ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Evaluación de algoritmos de visión estereoscópica para un sistema de reconocimiento de la mano

AUTOR: Emilio Alcantud Senent
DIRECTOR: Javier Toledo Moreo

11 / 2014

Autor	Emilio Alcantud Senent
E-mail del Autor	orion5ejmps@gmail.com
Director(es)	Javier Toledo Moreo
E-mail del Director	javier.toledo@upct.es
Codirector(es)	-----
Título del PFC	Evaluación de algoritmos de visión estereoscópica para un sistema de reconocimiento de la mano.
Descriptores	SAD, Census, Mapas de disparidad, Visión estereoscópica

Resumen

El objetivo del proyecto es la evaluación de una serie de algoritmos de visión estereoscópica destinados a la obtención de mapas de disparidad de cara a contemplar su posible aplicación sobre un algoritmo de reconocimiento de la mano.

En una primera fase, se realiza la implementación software de los algoritmos mediante Matlab, y se lleva a cabo su evaluación sobre una serie de imágenes prediseñadas ofreciendo un análisis comparativo de las diferentes soluciones en base a la aplicación de distintas configuraciones sobre los parámetros de cada uno de los algoritmos.

En una segunda fase, se realiza la toma de imágenes reales mediante un sistema de cámaras estéreo y se lleva a cabo la misma evaluación realizada en la primera fase para las imágenes prediseñadas, pero esta vez aplicando los algoritmos sobre las imágenes reales y específicas.

Finalmente se exponen las conclusiones alcanzadas tras el análisis de las diversas pruebas realizadas determinando cuáles han resultado ser las soluciones más apropiadas.

Titulación	Ingeniería de Telecomunicación
Intensificación	-------
Departamento	Dpto. de Electrónica, Tecnología de Computadoras y Proyectos
Fecha de Presentación	12- 2014

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

El objetivo del proyecto que aquí se presenta es el de llevar a cabo el análisis de un conjunto de algoritmos de correspondencia estéreo basados en área cuya función consiste en realizar el cálculo del mapa de disparidad de una imagen estereoscópica dada, obtenida a partir de un sistema de dos cámaras. Concretamente, la finalidad última en la que se enmarca el proyecto es la de contribuir a mejorar las prestaciones de un algoritmo para el reconocimiento de la mano en una imagen dada, destinado a un sistema de interacción hombre-computador que, particularmente, contempla el uso de la mano como "dispositivo" capaz de realizar acciones en aplicaciones ejecutadas en un ordenador u otro sistema de procesamiento. Por este motivo, debido a las características de la aplicación, la mano será considerada como el "objeto" más cercano en la escena de las imágenes estereoscópicas obtenidas y empleadas para el análisis. De esta forma, con la ayuda de la información proporcionada por el mapa de disparidad, se pretende mejorar la precisión y la robustez del sistema.

El proyecto queda dividido en 5 capítulos, siendo el primero de ellos el capítulo de introducción, en el que se detallan los objetivos del proyecto y el contexto en el que se enmarca, así como un breve resumen de la estructura y el contenido del presente informe. Además, este primer capítulo cuenta con un subapartado denominado *"ESTUDIO PREVIO"* en el que se expone el estado del arte en lo que a los algoritmos de correspondencia estéreo se refiere.

Tras este primer capítulo introductorio, se encuentra el capítulo 2 denominado *"ALGORITMOS IMPLEMENTADOS"*, en el cual se realiza una presentación y descripción detallada de los diferentes algoritmos para el cálculo de mapas de disparidad que se han implementado y empleado para llevar a cabo el análisis de las imágenes utilizadas en el proyecto, con la ayuda de los flujogramas correspondientes para cada uno de los algoritmos.

Seguidamente, en el capítulo 3 denominado *"RESULTADOS OBTENIDOS PARA LAS IMÁGENES ESTÁNDAR"*, se presentan detalladamente los resultados de los análisis y las pruebas realizadas con cada uno de los algoritmos descritos en el capítulo anterior, incluyendo imágenes, gráficas y tablas explicativas, para las imágenes prediseñadas de la universidad de *Middlebury*, que constituyen un estándar para la evaluación y diseño de algoritmos de visión estéreo.

Acto seguido, en el capítulo 4 *"RESULTADOS OBTENIDOS PARA LAS IMÁGENES REALES Y ESPECÍFICAS"* se muestran los resultados de los análisis y pruebas realizadas con el conjunto de imágenes específicas tomadas mediante el sistema *webcam* descrito en el propio capítulo.

Para finalizar, se exponen en el capítulo 5 *"CONCLUSIONES Y LÍNEAS FUTURAS"* las conclusiones alcanzadas tras la culminación del presente proyecto y el análisis de los resultados proporcionados por las pruebas realizadas durante el mismo mostrados en los capítulos anteriores del informe que aquí se detalla, así como también las posibles líneas de actuación futuras que quedan abiertas para futuros proyectos.

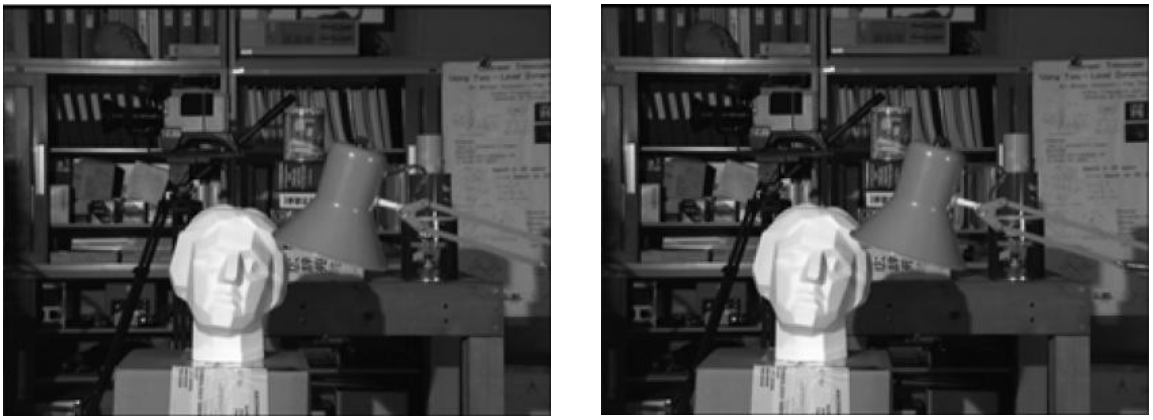
1.1. Estudio Previo

Se llama visión estéreo a la capacidad de recuperar la estructura tridimensional de una escena a partir de, por lo menos, dos vistas o imágenes diferentes de la misma. La estructura que se recupera es la posición de los objetos presentes en la escena, fundamentalmente la profundidad (distancia al observador) de los objetos. Desde hace muchos años el tema de la visión estéreo ha sido muy estudiado para la visión por computador, para aplicaciones tales como navegación de robots, creación de realidad virtual, seguimiento y vigilancia, etc. Sin embargo, lograr que una computadora "pueda ver" es un desafío que aún no ha sido resuelto completamente, pues existen varias dificultades que se plantean cuando se intenta abordar este problema utilizando una computadora como sistema de procesamiento, tales como la adquisición de las imágenes, el ruido presente en la misma, las diferencias de intensidad de un mismo punto en ambas imágenes, oclusiones, complejidad de la escena, etc. No obstante, a pesar de las dificultades, se han hecho grandes avances y se han logrado resultados importantes en diversas aplicaciones.

En los seres humanos, la percepción visual de la estructura tridimensional es realizada por el sistema visual humano, mediante los ojos y el cerebro. Dada la posición de los ojos, las imágenes que éstos presentan al cerebro son prácticamente idénticas salvo por una cierta diferencia en la posición relativa de los objetos en dichas imágenes. A esta diferencia relativa se le denomina disparidad, y tiene una relación directa con la distancia (profundidad) a la que se encuentran los objetos entre sí y respecto del observador. El cerebro es capaz de interpretar esa diferencia y reconstruir la estructura tridimensional de la escena que se está observando. El problema de la correspondencia estéreo trata por tanto de dar solución a ese mismo rompecabezas que el cerebro resuelve automáticamente de forma tan aparentemente sencilla. La formulación que se plantea para resolver el problema consiste en modelar el sistema de visión humano, utilizando para ello un par de cámaras (a efectos de ser los ojos del sistema) que obtienen el par estéreo de la escena, y una computadora o sistema de procesamiento que (a efectos de ser el cerebro del sistema) procede a realizar el análisis y tratamiento de las imágenes proporcionadas por las cámaras para calcular finalmente la geometría tridimensional de la escena.

Existen por tanto tres etapas bien diferenciadas en la recuperación de la estructura de una escena. En primer lugar, seleccionar un punto en una de las imágenes, después, encontrar ese mismo punto en la otra imagen del par estéreo, y por último,

medir la disparidad entre la posición de esos dos puntos. Se trata pues de localizar para cada punto de una de las imágenes del par, su correspondiente en la otra, y medir la diferencia relativa de posición entre ambos, generando así lo que se conoce como mapa de disparidad, una imagen de profundidades para cada uno de los puntos seleccionados de la escena proyectados en ambas fotografías. Este mapa consiste pues en una imagen (relativa a una de las del par estéreo), generalmente en escala de grises, en la cual el valor de cada píxel es el valor de disparidad asignado a cada punto de la imagen. Disparidades pequeñas refieren a objetos que están más al fondo en la escena, por lo que se muestran en el mapa en gris más oscuro, mientras que disparidades grandes refieren a objetos que están más cerca del observador, más en primer plano, por lo que se muestran en gris más claro. De esta forma, se van escalando los objetos de la escena en cuanto a tonalidad de gris, de manera que cuanto más claro es un objeto en la imagen del mapa de disparidad, significa que está más en primer plano en la escena, y cuanto más oscuro es el objeto, implica que está más al fondo. A continuación, se muestra un ejemplo de un mapa de disparidad en la *figura 1.1.2*, relativo a la imagen izquierda del par estéreo mostrado en la *figura 1.1.1*:



(a) (b)

Figura 1.1.1. Imagen izquierda de la escena (a), imagen derecha de la escena (b).

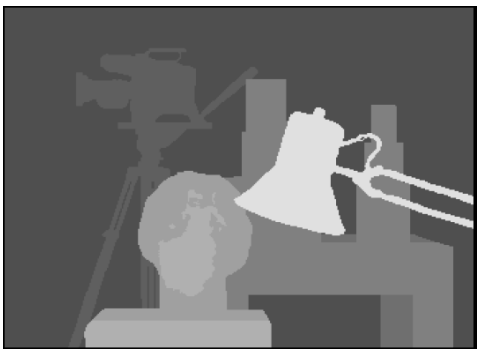


Figura 1.1.2. Mapa de disparidad correspondiente a la imagen izquierda de la *figura 1.1.1*.

A continuación se muestra gráficamente en la *figura1.1.3* la geometría y configuración del sistema convencional de visión estereoscópica que permite el cálculo de la disparidad:

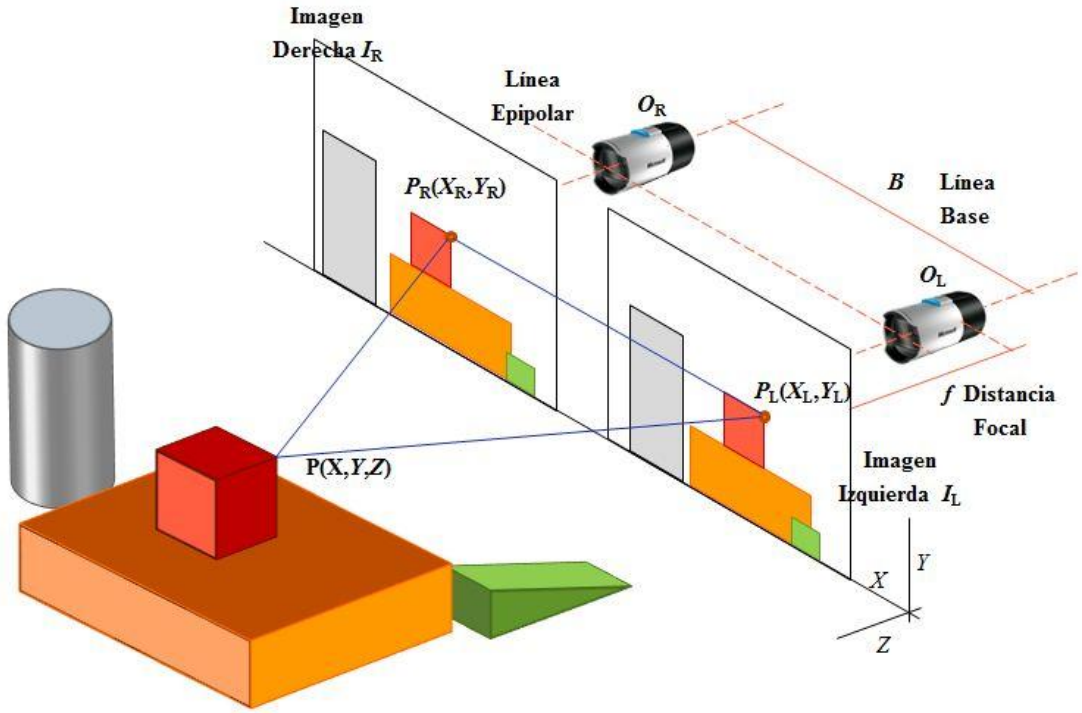


Figura 1.1.3. Geometría de un par de cámaras estéreo con ejes ópticos paralelos.

Dos cámaras forman un par estéreo, con sus ejes ópticos mutuamente paralelos. Ambas tienen la misma distancia focal *f*, con centros O_R y O_L separados una distancia *b* llamada línea de base, de manera que las imágenes que se forman, I_R e I_L, estén en planos paralelos. De este modo la línea de base es paralela a la coordenada *x* de las imágenes. En este modelo considerado, un punto en el espacio tridimensional *P* de coordenadas *(x,y,z)* se proyecta en cada una de las imágenes bidimensionales en los puntos P_L y P_R con coordenadas (x_L,y_L) y (x_R,y_R), respectivamente. El plano que contiene a los puntos *P*, O_R y O_L, intersecta a las imágenes en una recta denominada *línea epipolar*, de forma que, un punto P_L en dicha línea de la imagen I_L, tiene su correspondiente en algún punto de la misma línea en la imagen I_R. De esta forma, puesto que el desplazamiento entre los centros ópticos de las dos cámaras es exclusivamente horizontal, la posición de los puntos correspondientes entre las dos imágenes solamente puede diferir en la componente horizontal, reduciendo así la búsqueda del correspondiente a un punto P_L en una línea de la imagen I_L, a la misma línea en la imagen I_R. Esto es lo que se conoce como *restricción de epipolaridad*.

En la siguiente figura se puede ver cómo se relacionan los parámetros definidos en el par estéreo que permiten obtener la relación entre la disparidad *d* y la profundidad *Z* del punto *P*:

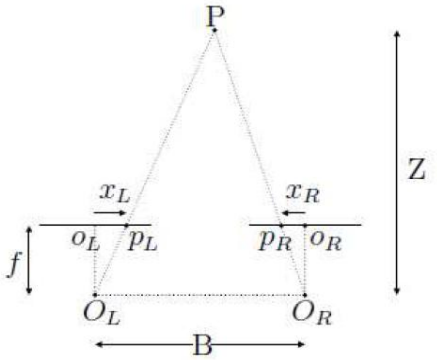


Figura 1.1.4. Esquema de disparidad.

La disparidad es la diferencia en las coordenadas horizontales de los puntos P_L y P_R, es decir, $d=X_L-X_R$. Las coordenadas de P_L y P_R quedan relacionadas mediante:

$$\begin{Bmatrix} x_L = x_R + d \\ y_L = y_R \end{Bmatrix}$$

Considerando los triángulos P-O_L-O_R, p_R-o_R-O_R y p_L-o_L-O_L, utilizando semejanza entre triángulos se llega a:

$$d = \frac{f}{Z} B$$

Entonces, se tiene la relación entre *d* y *Z*:

$$d \propto \frac{1}{Z}$$

En base a esta última ecuación, se puede recuperar, salvo por una constante de escala, la profundidad de cada píxel en cada una de las imágenes a partir de la disparidad calculada.

Los algoritmos destinados a obtener el mapa de disparidad de una imagen, se denominan algoritmos de correspondencia estéreo, y pueden ser clasificados, fundamentalmente, en dos grandes grupos:

- Basados en área
- Basados en características

Los algoritmos basados en área realizan la comparación mediante ventanas de la imagen de dimensión fija, centradas en el punto a analizar. Para cada pixel de una de las

imágenes del par estéreo, se calcula la correlación de intensidades de la ventana centrada en dicho píxel, y una ventana del mismo tamaño centrada en cada uno de los píxeles a analizar de la otra imagen para una región seleccionada de la misma. La correspondencia entre los puntos queda determinada por la ventana que minimiza un cierto criterio de correlación dentro de la región de búsqueda establecida. Aquí se pueden distinguir métodos como la suma de diferencias absolutas (SAD), la suma de diferencias al cuadrado (SSD) o la correlación cruzada normalizada (NCC), en función del criterio de correlación empleado. La ventaja de estos algoritmos es que obtienen buenos resultados para imágenes con textura visual importante (imágenes donde los objetos que aparecen en ellas poseen una gran cantidad de rasgos visuales en su superficie, producto del comportamiento de la luz sobre la superficie del objeto), permiten crear mapas densos de disparidad y son fáciles de paralelizar. En la siguiente figura se muestra de forma gráfica el proceso de búsqueda para este tipo de algoritmos:

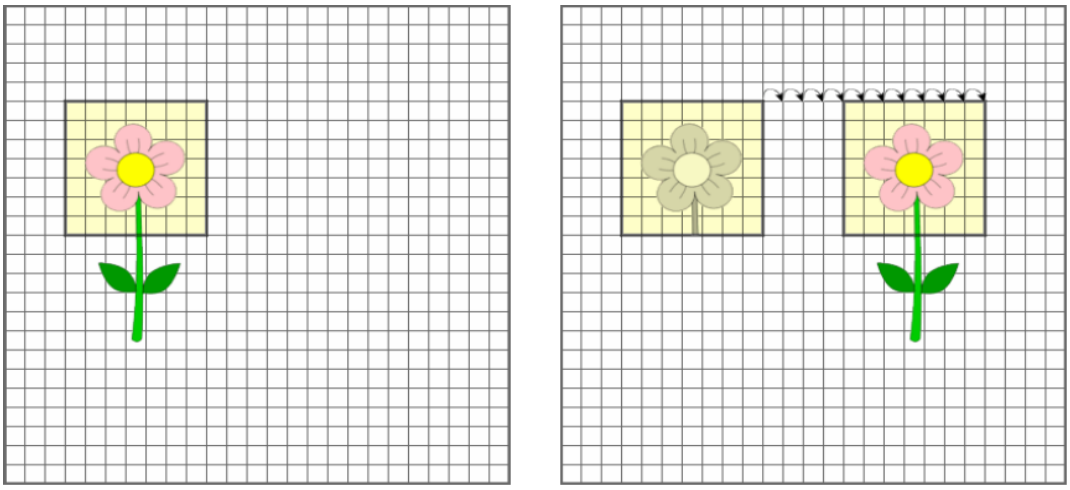
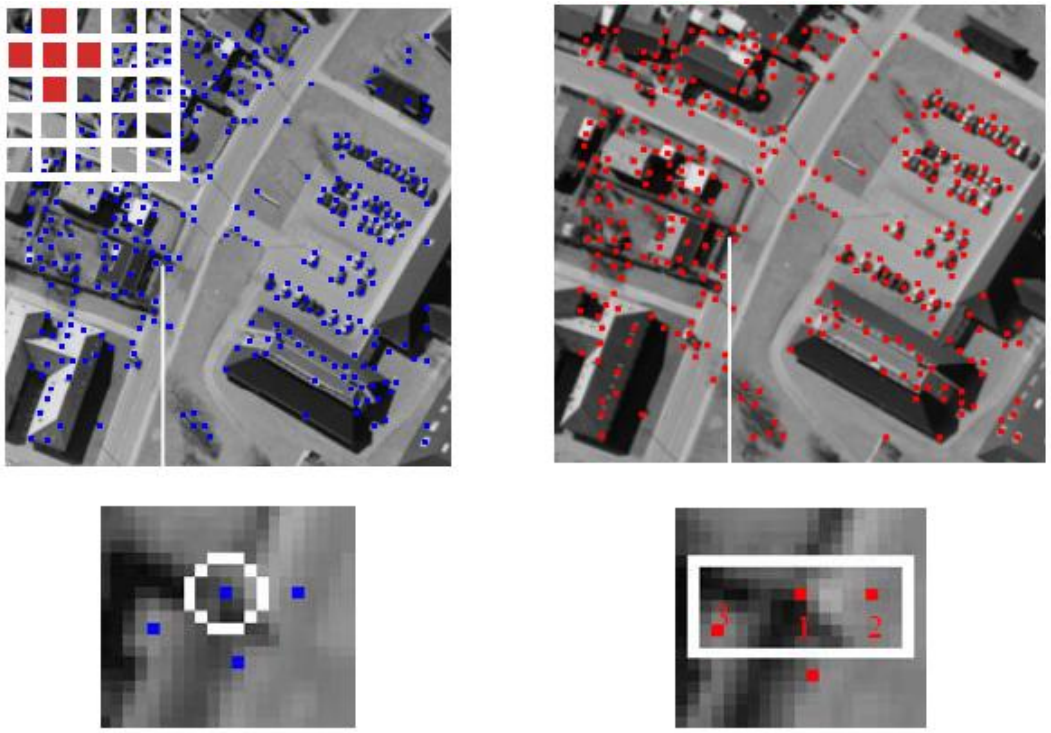


Figura 1.1.5. Proceso de búsqueda de correspondencia basada en área en un par de imágenes estéreo, consistente en una imagen izquierda (a) y una imagen derecha (b).

Por otra parte, los algoritmos basados en características, en lugar de realizar la correspondencia píxel a píxel, la realizan para determinadas estructuras de un nivel superior al píxel. Concretamente, como características se entienden puntos de borde (tales como esquinas), segmentos de borde (rectos o curvilíneos, tales como aristas) y regiones específicas de alguna parte de la imagen. Para obtenerla correspondencia mediante este tipo de algoritmos, se realiza la comparación entre ciertos atributos definidos para cada una de las características mencionadas, resultando como correspondientes aquellas cuyos atributos sean lo más parecidos entre sí. A continuación la *figura 1.1.6* muestra gráficamente esta metodología:



(a) (b)

Figura 1.1.6. Proceso de búsqueda de correspondencia basada en características en un par de imágenes estéreo, consistente en una imagen izquierda (a) y una imagen derecha (b).

En el par estereoscópico aparecen los puntos de interés extraídos, que en este caso corresponden a puntos de borde. Supongamos que se desea encontrar la correspondencia para el punto de borde remarcado en el interior del círculo de la zona ampliada de la imagen izquierda. En primer lugar se determina la ventana de búsqueda en la imagen derecha (la cual se muestra en la zona ampliada de la imagen derecha), en la que se encuentran varios puntos de interés (numerados del 1 al 3 para que se aprecien). A continuación, se va centrando en estos tres puntos la ventana de correspondencia y se calculan los coeficientes de correlación, resultando como correspondiente aquel punto cuyos atributos son los que más se asemejan a los del punto que se desea corresponder. Algunos de los atributos para un punto de borde pueden ser el signo, la orientación y la intensidad. En el caso en que las características extraídas fueran líneas, se podrían evaluar propiedades como su longitud, orientación, las coordenadas del punto medio o la media de intensidad en la línea.

Para el desarrollo del presente proyecto se ha optado por el empleo de los algoritmos basados en área, cuyas características son más apropiadas para la tarea que aquí se propone, ya que obtienen un valor de disparidad para cada píxel de la imagen, generando así un mapa de disparidad denso y teniendo en cuenta por otra parte, que los métodos basados en características no son apropiados para emplearlos en aplicaciones que deben funcionar en tiempo real, ya que demandan un mayor tiempo de

procesamiento debido a las complejas operaciones que necesitan llevar a cabo previamente a la obtención de la disparidad para extraer las características de cada imagen. A continuación, en el capítulo 2, se procede a describir detalladamente cada uno de los algoritmos empleados.

CAPÍTULO 2. ALGORITMOS IMPLEMENTADOS

La realización del presente proyecto se centra en los algoritmos de correspondencia estéreo basados en área, tal y como se explicó en el capítulo *Introducción*. Por la rapidez que ofrece para la programación y evaluación de los resultados, *Matlab* ha sido la plataforma utilizada para la implementación de los algoritmos. En este capítulo se describen con detalle cada uno de ellos. A continuación, en la *figura 2.1* se muestra un diagrama que resume de forma gráfica las implementaciones realizadas y cómo se han agrupado para ser detalladas en el presente capítulo:

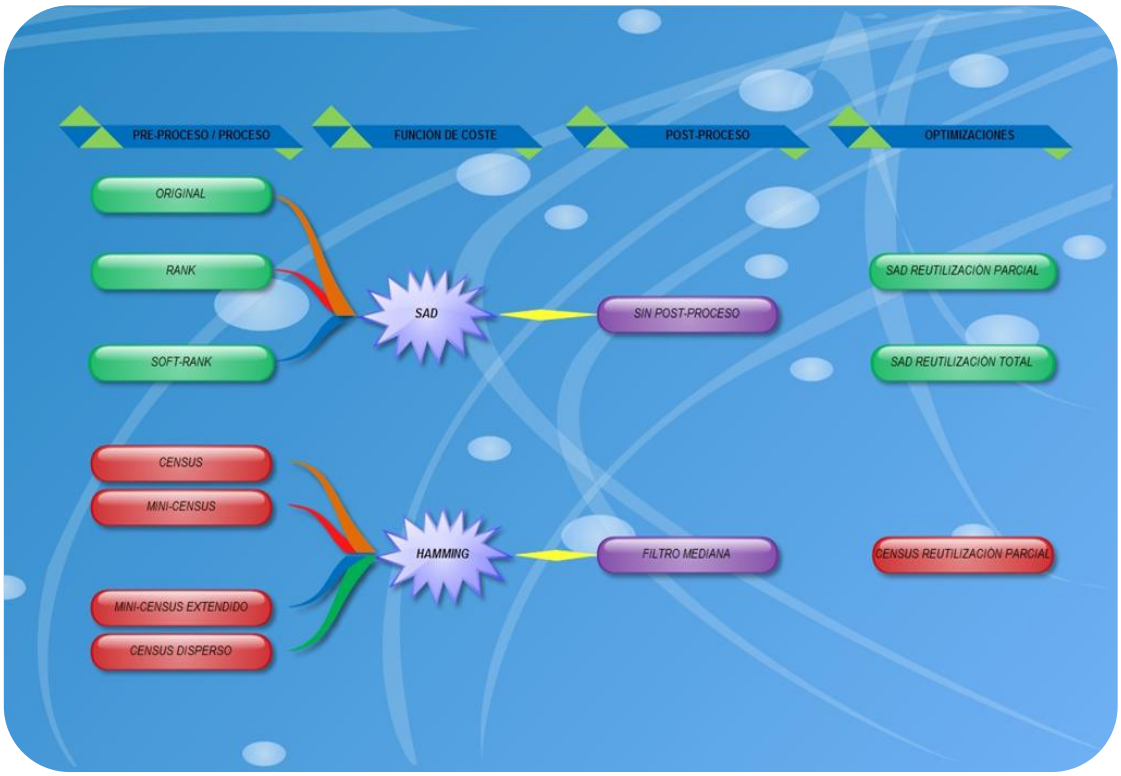


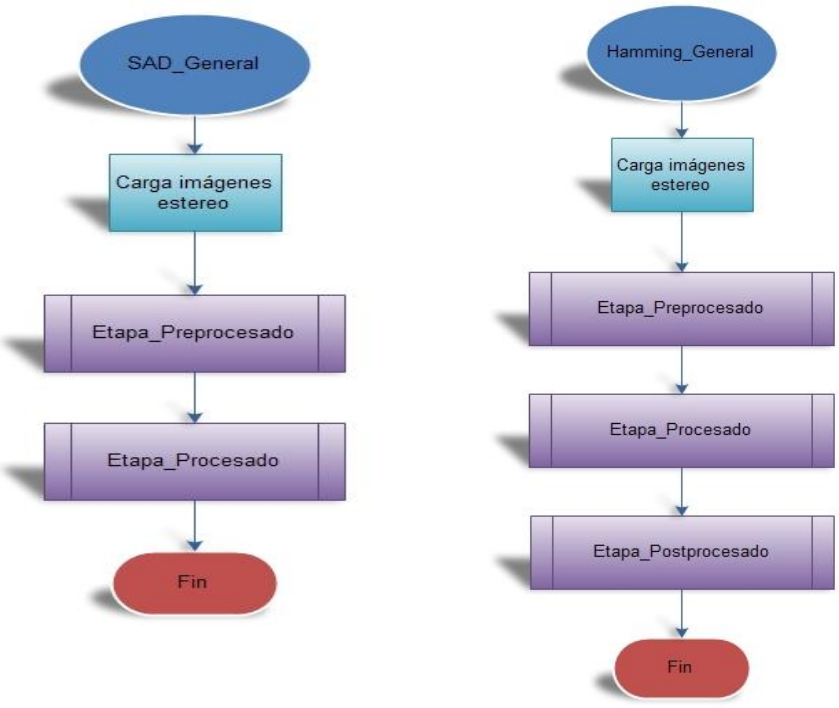
Figura 2.1. Diagrama resumen de las implementaciones realizadas.

Tal y como se muestra en el diagrama, las diferentes implementaciones se agrupan según la función de coste de la que hacen uso para llevar a cabo el cálculo de la disparidad, existiendo fundamentalmente dos, *SAD* y *Hamming*, que constituyen los dos apartados principales del presente capítulo.

En el primer apartado se detalla el algoritmo *SAD (Suma de diferencias absolutas)* en su implementación básica sobre las imágenes originales. En este apartado se incluyen las descripciones de las implementaciones *Rank* y *Soft-Rank*, que hacen uso de una transformación previa al uso del criterio *SAD* para llevar a cabo el cálculo de la disparidad. Además, se detallan en el subapartado 2.1.1 “Optimización por

reutilización" las implementaciones del algoritmo que se han llevado a cabo empleando la técnica de reutilización de valores durante el proceso de cálculo de la disparidad con la intención de disminuir el tiempo de cómputo del algoritmo, las cuales se han denominado *SAD con reutilización parcial* y *SAD con reutilización total*. El segundo apartado está dedicado al conjunto de implementaciones correspondientes al modelo de algoritmo que emplea el cálculo de las distancias de *Hamming* para obtener la disparidad, como son *Census* en su versión completa, *mini-Census*, *mini-Census extendido*, y una última variación a la que se ha denominado *Census disperso*, las cuales hacen uso, al igual que en el caso de las implementaciones *Rank* y *Soft-Rank* para el algoritmo *SAD*, de una transformada previa al uso del criterio *Hamming* para llevar a cabo el cálculo de la disparidad. Se detalla además en este apartado, al igual que se ha realizado para el algoritmo de *SAD*, la variación con reutilización de valores para el caso de *Census*, denominada *Census con reutilización parcial*, así como la implementación de un filtro de mediana con el objeto de emplearlo posteriormente al proceso de cálculo de la disparidad con la intención de lograr una mejora en la tasa de acierto del mapa resultante.

Como ya adelanta el diagrama, todos esos algoritmos mencionados y presentados en este capítulo están constituidos por un flujo general de funcionamiento similar. Dicho flujo se compone de 2 etapas bien diferenciadas, cada una de las cuales lleva a cabo una parte del proceso en la obtención del mapa de disparidad. En primer lugar, hay una etapa de pre-procesado en la cual se realizan las transformaciones y tratamiento previo sobre las imágenes estereoscópicas originales de la escena necesarios antes de realizar el cálculo de la disparidad. Acto seguido se lleva a cabo la etapa de procesado, que se encarga de implementar la obtención del mapa de disparidad según los cálculos requeridos por la función de coste propia de cada algoritmo. Para el caso de Hamming se concluye además con una tercera etapa de post-procesado en la que se realiza el tratamiento posterior al cálculo del mapa de disparidad para mejorar la tasa de acierto del mismo mendiante el empleo de un filtro de mediana. A continuación, en la *figura 2.2* se muestra gráficamente este flujo general de funcionamiento descrito:



(a) (b)

Figura 2.2. Flujograma *SAD* general (a), flujograma *Hamming* general (b).

A continuación se muestra además, en la *figura 2.3*, el flujograma correspondiente a la etapa de pre-procesado, ya que también es genérico para todas las implementaciones realizadas:

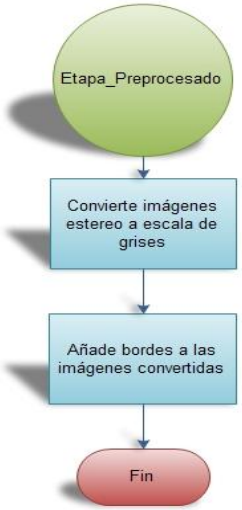


Figura 2.3. Flujograma Etapa Pre-pocesado.

En esta etapa, se convierten las imágenes del par estéreo originales a escala de grises, y se les añade bordes superiores, inferiores, a izquierda y a derecha (de valor nulo) en función del tamaño de la ventana de correspondencia elegido para prevenir la situación que se produce cuando la ventana de correspondencia se centra en los píxeles de los márgenes, en la que quedarían fuera de la imagen varios de los píxeles de la ventana, de esta manera se salva dicha situación. La función finalmente devuelve las imágenes convertidas a escala de grises con los bordes añadidos.

Cabe mencionar también que todos los algoritmos presentados en este capítulo hacen uso de la denominada *restricción epipolar*, mediante la cual se reduce la región de búsqueda para la correspondencia de un determinado píxel a una sola línea de la imagen, ya que el concepto de *epipolaridad* implica que la proyección de un píxel perteneciente a una línea determinada de una de las imágenes del par estará necesariamente sobre la misma línea de la otra imagen. La búsqueda de correspondencia se limita además a un rango máximo denominado *máxima disparidad*. De esta forma, la búsqueda de la correspondencia para un píxel determinado de la imagen izquierda se realiza únicamente sobre la misma línea en la imagen derecha desde la posición *(x,y)* hasta la posición *(x+dmax,y)*, siendo *dmax* la máxima disparidad aceptada. Así, mediante la *restricción de epipolaridad* y una elección adecuada del valor del parámetro de *máxima disparidad*, se reduce el coste computacional necesario para llevar a cabo el cálculo del mapa de profundidad.

Se procede a continuación a la descripción de los algoritmos implementados.

2.1 Algoritmos basados en el criterio *SAD*

En este apartado se describen las implementaciones realizadas para el algoritmo de *SAD*. Dicho algoritmo emplea como función de coste para el cálculo de la similitud entre parejas de ventanas, a la hora de obtener la disparidad, la suma de las diferencias absolutas entre los valores que componen las ventanas. En primer lugar, se ha realizado la implementación básica del algoritmo. Dicho algoritmo comienza con la etapa de pre-procesado, cuyo flujograma ya se ha detallado en la figura 2.3.

Seguidamente, se realiza la etapa de procesado, cuyo flujograma se muestra en la *figura 2.1.1*. En esta etapa, se realiza el cálculo de la disparidad según la función de coste definida por el algoritmo *SAD*. Dicha función de coste queda definida por la siguiente ecuación:

$$C_{SAD} = \sum_u \sum_v |I_d(x+u, y+v) - I_i(x+u+d, y+v)|$$

En la fórmula, I_d e I_i representan las intensidades para las imágenes derecha e izquierda respectivamente, *(x,y)* es el punto central de la primera imagen, *(x+d,y)* es un

punto de la línea correspondiente en la otra imagen desplazado una distancia *d*, *u* es el índice que recorre la ventana en dirección horizontal y *v* es el índice que recorre laventana en dirección vertical. Se asume que la ventana de mayor similitud es aquella con un valor menor de C_{SAD} entre todas aquellas que se pueden formar en el rango d_{max}.

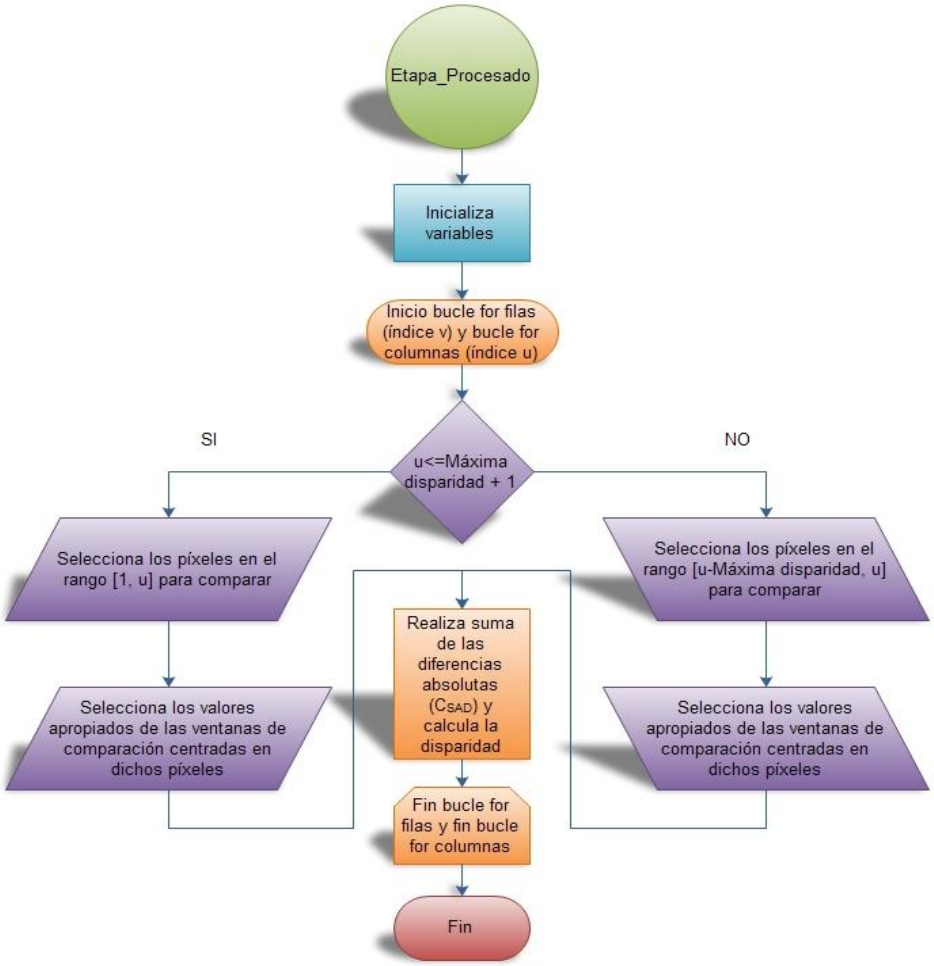


Figura 2.1.1. Flujograma etapa de procesado.

La *figura 2.1.1* describe a grandes rasgos el proceso implementado para realizar este cálculo, que se inicia preparando las variables que van a ser necesarias para realizar las operaciones, que son dos matrices que contendrán las dos ventanas que se comparen en cada momento y una variable que almacena el tamaño de la ventana de coste. Comienzan entonces los bucles anidados que recorrerán las imágenes de arriba abajo (es decir, por filas, mediante la variable *v*) y de izquierda a derecha (es decir, por columnas, mediante la variable *u*) y dependiendo de en qué columna (valor dado por *u*) se encuentre el píxel para el que se va a realizar el cálculo de la correspondencia, se selecciona el rango de píxeles apropiado donde se buscará su correspondiente. Si el píxel para el que se va a calcular la correspondencia se encuentra en una posición (en el eje *x*, el eje de las columnas, indicado por la variable *u*) inferior a la indicada por el valor de Máxima disparidad + 1, se seleccionarán para realizar la comparación los píxeles que se encuentren en el rango [1, *u*], mientras que si el valor de u ha superado ya el umbral de Máxima disparidad + 1, se seleccionarán los píxeles que se encuentren en

el rango [*u* – Máxima disparidad, *u*]. Por ejemplo, estando en una fila cualquiera de las imágenes, la fila 1 por ejemplo (*v*=1), si el valor de Máxima disparidad fuera 3, y el píxel para el que se desea buscar su correspondiente es el segundo de la fila (*u*=2), se buscará su correspondiente en el rango [1, 2], de forma que se comparará con el píxel 1 o con el que corresponde a su misma posición en la imagen derecha, mientras que, si el píxel fuera el 5 (*u*=5), el rango de píxeles en el que se buscaría su correspondiente sería el rango [2, 5], respetando así el máximo de disparidad permitido, que para el caso de este sencillo ejemplo establece que la distancia máxima entre la posición de un píxel y su correspondiente en la otra imagen no debe superar 3 píxeles. El hecho de que solamente se compare con valores que se encuentran a la izquierda del píxel en cuestión, se debe a que para este proyecto los algoritmos se han implementado de forma que se hagan corresponder píxeles de la imagen izquierda en la imagen derecha, con lo cual carece de sentido buscar correspondencias a la derecha del píxel en cuestión ya que en este caso tan solo pueden estar a su izquierda. Una vez definido el conjunto de píxeles a comparar, se seleccionan los valores apropiados que componen las ventanas de comparación que se centran en dichos píxeles y se procede a compararlas mediante el sumatorio de las diferencias absolutas de sus valores tal y como indica el criterio *SAD*, devolviendo así finalmente como resultado el mapa de disparidad.

Hasta aquí la implementación básica del algoritmo. No obstante, para el presente proyecto, se han llevado a cabo algunas variaciones de esta implementación, como la denominada *Rank.* El algoritmo *Rank* consiste en la aplicación de una transformación sobre el par de imágenes estereoscópicas de la escena antes de proceder al cálculo de la correspondencia mediante alguna función de coste, generalmente la de *SAD* [14]. Dicha transformación se denomina *transformada Rank* y se realiza para una determinada ventana rectangular centrada en cada uno de los píxeles del par estereoscópico. Al realizar la transformada, se sustituye el valor de intensidad del píxel sobre el que se centra la ventana de transformación por el rango (o posición) que ocupa dicho píxel (en lo que a valor de intensidad se refiere) con respecto al resto de píxeles dentro de la misma. De esta forma, puesto que el coste depende únicamente del orden de intensidades y no de la magnitud, se pueden tolerar todas las distorsiones radiométricas que preserven ese orden, incrementando así la robustez ante cambios de tonalidad y luminosidad. La función que define la *transformada Rank* es la siguiente:

$$I_{Rank}(\boldsymbol{p}) = \sum_{q \in N_p} T[I(\boldsymbol{q}) < I(\boldsymbol{p})]$$

Según esta función, fundamentalmente, el valor que toma el píxel sobre el que se realiza la *trasnformada Rank* se define como el número de píxeles de la ventana para los cuales su intensidad es menor que la intensidad de dicho píxel central. En la *figura 2.1.2* se muestra un ejemplo gráfico con una ventana de dimensión *5x5*, donde la intensidad del píxel central es de 27. Tras aplicar la *transformada Rank*, la información de la

ventana se codifica en el dígito 5, puesto que hay 5 píxeles cuya intensidad es menor que la del píxel central.

81	47	90	58	12
70	91	34	63	24
09	75	27	85	54
69	95	75	96	49
15	76	97	06	95

$\rightarrow$ 5

Figura 2.1.2. *Transformada Rank* para una ventana de dimensión *5x5*.

La etapa de procesado es la ya mostrada en la *figura 2.1.1* para la implementación básica del algoritmo, puesto que en este caso el algoritmo *Rank* hace uso del criterio *SAD* para el cálculo de la correspondencia. La diferencia radica en la etapa de pre-procesado en la que, tras haber convertido las imágenes del par estéreo a escala de grises y haberles añadido los bordes, se añade también el proceso que implementa la *transformada Rank* sobre ellas.

Sin embargo, la *transformada Rank* es susceptible al ruido en áreas con poca textura, tal y como se puede apreciar en la zona de la imagen que hay a la derecha del oso que se muestra en la siguiente figura:

(a) (b)

Figura 2.1.3. Imagen original en escala de grises (a), resultado de la *transformada Rank* sobre la imagen original (b).

Para reducir este problema, se propuso una variación de este algoritmo denominada *Soft-Rank* [14], que define una zona de transición lineal suave entre 0 y 1

para valores que están muy próximos. La función que define dicha transformación es la siguiente:

$$I_{SoftRank}(\boldsymbol{p}) = \sum_{q \in N_p} \min\,(1, \max\,(0, \frac{I(\boldsymbol{p}) - I(\boldsymbol{q})}{2t} + \frac{1}{2}))$$

Se puede apreciar en la siguiente figura como al emplear esta transformación el resultado es mucho menos ruidoso en áreas con escasa textura:

(a) (b)

Figura 2.1.4. Imagen original en escala de grises (a), resultado de la *transformada Soft-Rank* sobre la imagen original (b).

De nuevo, al igual que ocurre con Rank, la diferencia en el flujo de programa de esta implementación con respecto a la implementación básica radica en la etapa de pre-procesado en la que se incluye la implementación del proceso que realiza la *transformada Soft-Rank* sobre las imágenes del par convertidas a escala de grises una vez añadidos los bordes.

2.1.1 Optimización por reutilización

Se han llevado a cabo además otras dos implementaciones alternativas para reducir el tiempo de ejecución que conlleva la implementación directa del algoritmo *SAD*, que consisten en la reutilización de los resultados parciales [13] de las operaciones que se van obteniendo a lo largo de las distintas iteraciones que se llevan a cabo hasta completar el algoritmo.

Si el flujo del algoritmo se encuentra en una fila determinada de las imágenes, donde éstas se procesan de izquierda a derecha hasta concluir todos los píxeles de la fila para poder pasar a la siguiente, se puede obtener el valor de *SAD* correspondiente a un determinado píxel de la fila para un determinado valor de disparidad, reutilizando el valor de *SAD* que se obtuvo para la posición del píxel anterior de esa misma fila para esa misma disparidad, añadiendo a ese valor el resultado *SAD* de una columna a la

derecha, y restando el resultado *SAD* de una columna a la izquierda, ahorrando de esta forma el cómputo del resto de columnas que forman la ventana de coste, puesto que ya se ha llevado a cabo en la posición anterior. La *figura 2.1.1.1* muestra gráficamente dicho proceso:

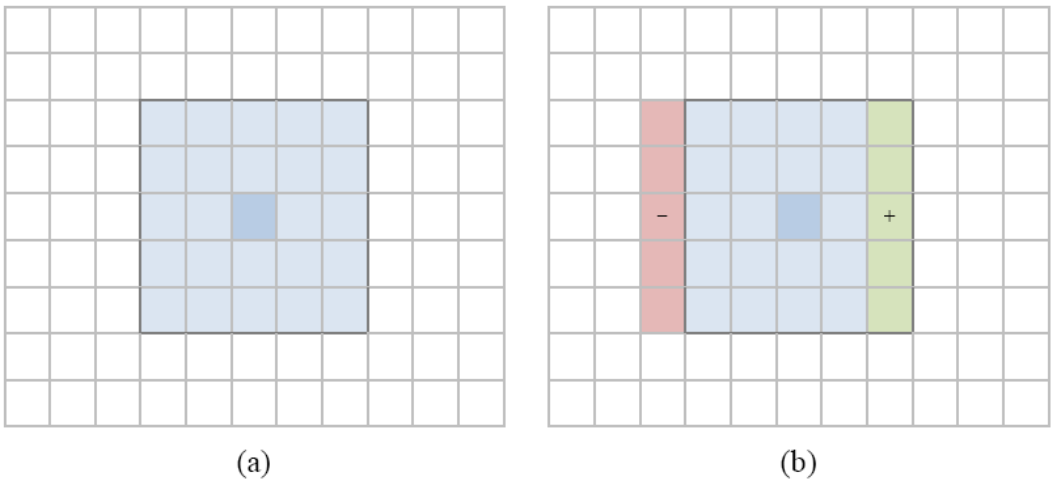


Figura 2.1.1.1. Método directo para el cálculo de *SAD* (a) y método alternativo que reutiliza las columnas (b).

Este concepto de reutilización parcial por columnas también puede ser aplicado a las filas, cuando se pasa de una fila a la siguiente. La *figura 2.1.1.2* lo muestra gráficamente para una sola columna de una supuesta ventana de coste:

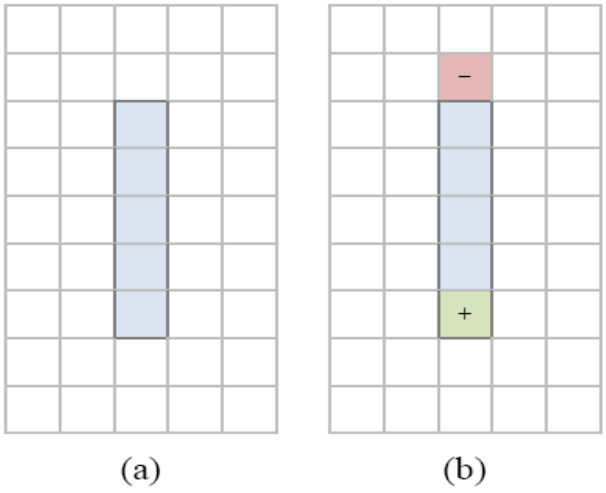


Figura 2.1.1.2. Método directo para el cálculo de *SAD* (a) y método alternativo que reutiliza los valores de la fila anterior (b).

Para comprobar este hecho matemáticamente, se realiza el siguiente ejemplo. Supóngase que se tienen las dos imágenes del par estéreo denominadas I_i e I_d a partir de las cuales se desea obtener el mapa de disparidad mediante el algoritmo *SAD*. Con el objeto de simplificar los cálculos, se establece un tamaño de ventana de coste de 3x3, y se fija el parámetro de máxima disparidad a 1. Suponiendo que el origen de coordenadas

está en *(x=0, y=0)*, en las coordenadas *(x=2, y=2)*, siendo *d*el valor de disparidad que se está comprobando, y *AD* la diferencia absoluta para una posición determinada, se tiene:

$$SAD(x=2,y=2,d=0) = AD(1,1,0)+AD(2,1,0)+AD(3,1,0)+AD(1,2,0)+AD(2,2,0)+AD(3,2,0)+AD(1,3,0)+AD(2,3,0)+AD(3,3,0)$$

$$SAD(x=2,y=2,d=1) = AD(1,1,1)+AD(2,1,1)+AD(3,1,1)+AD(1,2,1)+AD(2,2,1)+AD(3,2,1)+AD(1,3,1)+AD(2,3,1)+AD(3,3,1)$$

Para un incremento unidad en el eje horizontal respecto al origen *(x=3, y=2)*:

$$SAD(x=3,y=2,d=0) = AD(2,1,0)+AD(3,1,0)+AD(4,1,0)+AD(2,2,0)+AD(3,2,0)+AD(4,2,0)+AD(2,3,0)+AD(3,3,0)+AD(4,3,0)$$

$$SAD(x=3,y=2,d=1) = AD(2,1,1)+AD(3,1,1)+AD(4,1,1)+AD(2,2,1)+AD(3,2,1)+AD(4,2,1)+AD(2,3,1)+AD(3,3,1)+AD(4,3,1)$$

Se han marcado en color azul los términos comunes para la posición anterior *(x=2, y=2)* para cada valor de disparidad. Los términos no comunes corresponden a una columna de diferencias absolutas situada en el borde derecho de la ventana. Aprovechando los términos comunes se puede calcular la suma de diferencias absolutas reutilizando las sumas de diferencias absolutas de columnas de iteraciones anteriores.

Así mismo, continuando con el ejemplo propuesto, para un incremento en el eje vertical con respecto al origen *(x=2, y=3)* se tiene:

$$SAD(x=2,y=3,d=0) = AD(2,2,0)+AD(3,2,0)+AD(4,2,0)+AD(2,3,0)+AD(3,3,0)+AD(4,3,0)+AD(2,4,0)+AD(3,4,0)+AD(4,4,0)$$

$$SAD(x=2,y=2,d=1) = AD(2,2,1)+AD(3,2,1)+AD(4,2,1)+AD(2,3,1)+AD(3,3,1)+AD(4,3,1)+AD(2,4,1)+AD(3,4,1)+AD(4,4,1)$$

De nuevo, se han marcado en color azul los términos comunes para la posición original *(x=2, y=2)* para cada valor de disparidad. Si ambas ideas se compaginasen, suponiendo ahora un escaneo horizontal, tan solo habría un único valor que no se habría calculado en iteraciones anteriores, que correspondería al valor *AD(4,4,1)*. Aplicando esta idea, se puede por tanto calcular la suma de diferencias absolutas en esta iteración

reutilizando las sumas de diferencias absolutas individuales de iteraciones anteriores, teniendo solamente que calcular una única diferencia absoluta. Este concepto en el que se reutilizan todos los resultados parciales posibles se denomina *reutilización total*.

Para el presente proyecto, se ha implementado una versión del algoritmo *SAD con reutilización parcial* por columnas y otra versión de *SAD con reutilización total*. A continuación se muestran los flujogramas correspondientes a las etapas de procesado de dichos algoritmos, ya que la etapa de pre-procesado se mantiene invariante con respecto a la versión original del algoritmo.

La *figura 2.1.1.3* muestra el diagrama correspondiente a la etapa de procesado para la versión de *SAD con reutilización parcial*. El proceso comienza preparando las variables que serán necesarias para realizar las operaciones, tras lo cual se inician los bucles que recorren las imágenes de arriba hacia abajo (mediante la variable *v*) y de izquierda a derecha (mediante la variable *u*) y, siempre que el píxel a corresponder sea el primero de cada una de las filas de la imagen (*u=1*), la disparidad para dicho píxel se establece como nula directamente, ya que en este caso únicamente se compara un solo par de ventanas, porque, como se explicó en el apartado 2.1 para la implementación básica del algoritmo, tan solo se buscan las correspondencias a la izquierda del píxel en cuestión, y por tanto la disparidad siempre resultará nula para los píxeles que ocupan este lugar (ya que no existen píxeles a su izquierda y por tanto solo puede compararse consigo mismo), habiendo calculado y almacenado previamente los valores que será necesario reutilizar en las iteraciones posteriores. Por otra parte, si el flujo del programa se sitúa en cualquier otro píxel de la fila que no sea el primero, al igual que ocurría en la implementación básica del algoritmo, dependiendo de en qué columna (valor dado por *u*) se encuentre dicho píxel se selecciona el rango de píxeles apropiado para realizar la comparación (rango [1, *u*] si u está por debajo del umbral indicado por el valor de máxima disparidad + 1, o bien rango [*u* – máxima disparidad, *u*] si *u* está por encima de dicho umbral) y los valores de las ventanas centradas en dichos píxeles que se deben comparar, y una vez seleccionados y compuestas las ventanas, se calculan y almacenan los resultados que será necesario reutilizar en las iteraciones posteriores y se procede con la comparación mediante el sumatorio de las diferencias absolutas, reutilizando los valores almacenados de las iteraciones anteriores, acelerándose de esta manera el proceso de cálculo.

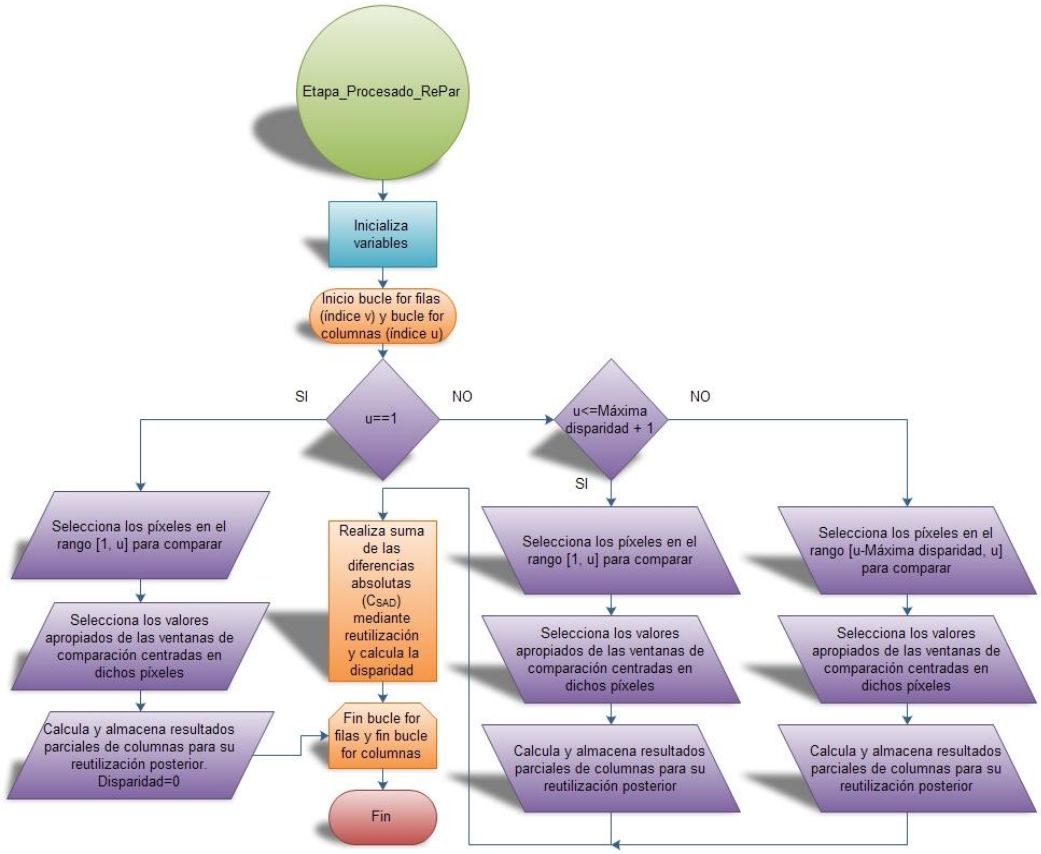
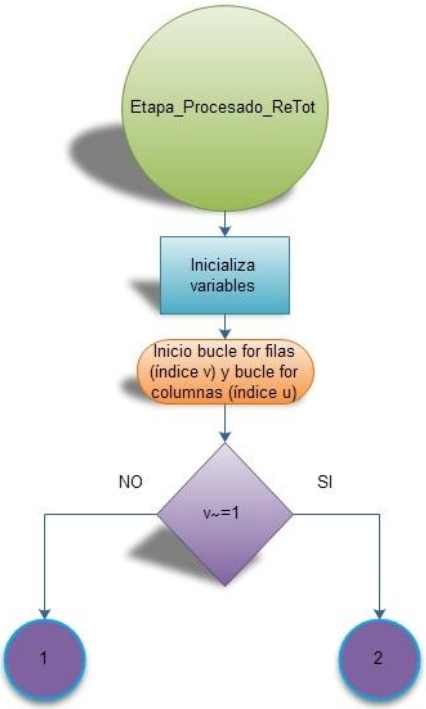
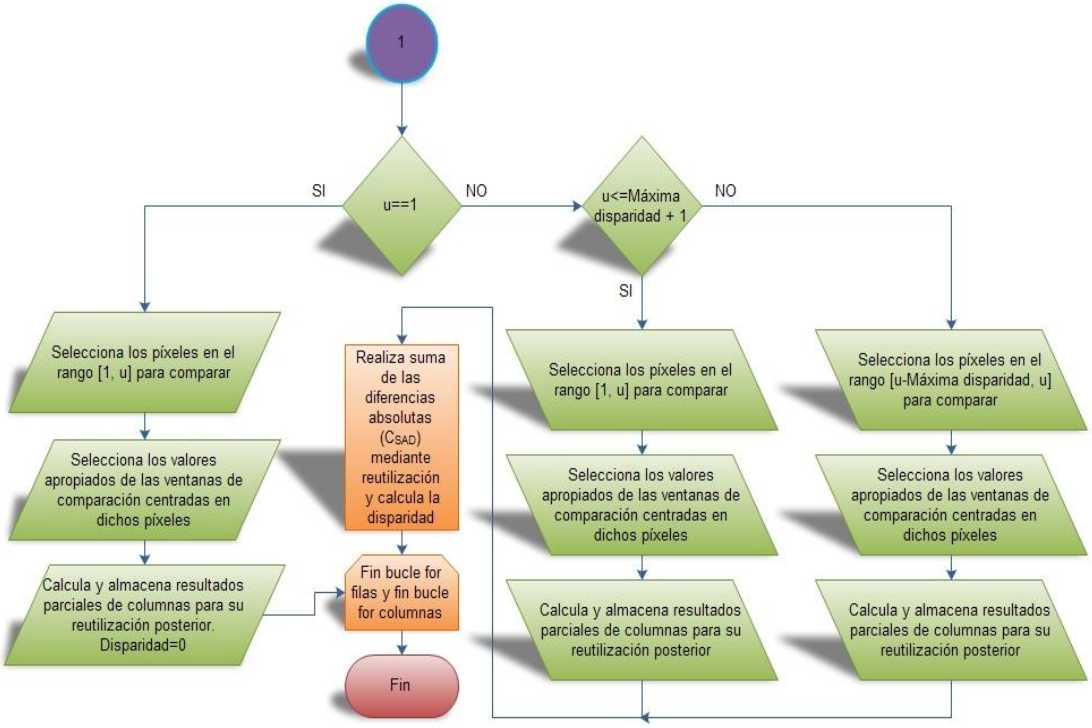


Figura 2.1.1.3. Flujograma etapa procesado *SAD con reutilización parcial* por columnas.

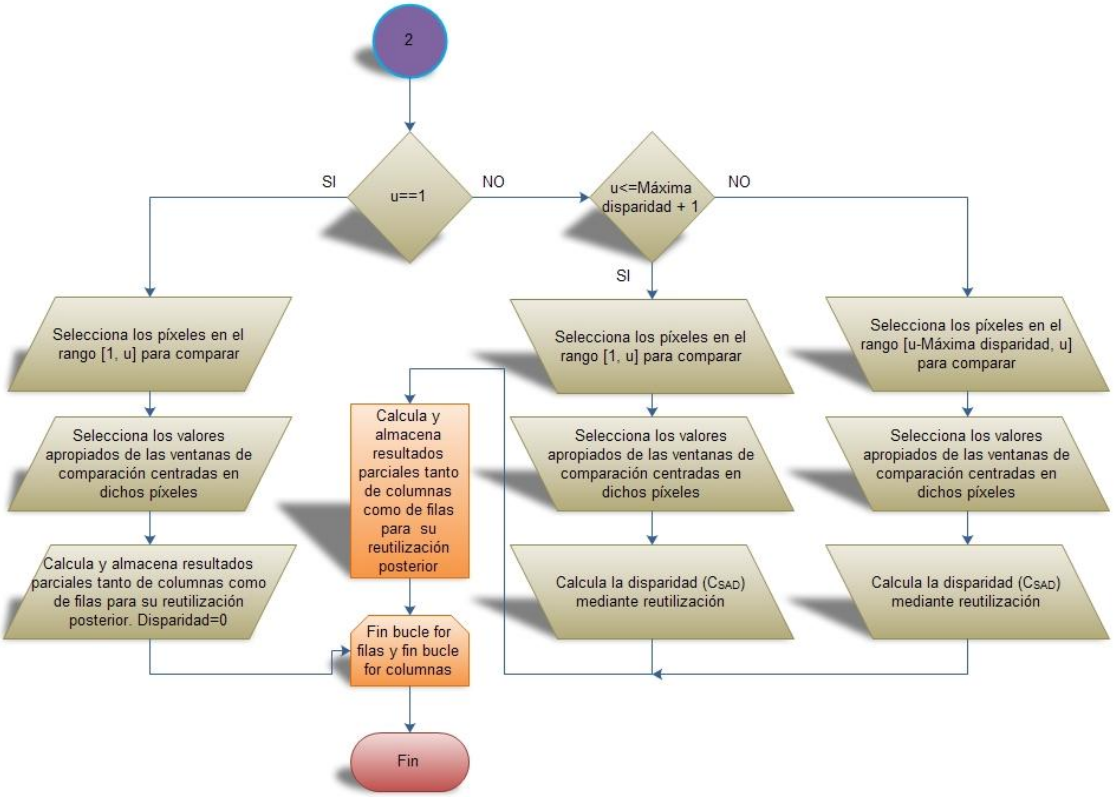
Para el caso de la versión de *SAD con reutilización total*, cuyo diagrama de flujo correspondiente a su etapa de procesado se muestra en la *figura 2.1.1.4*, tras haber preparado las variables necesarias para realizar las operaciones, y haber iniciado los bucles que recorren las imágenes, se comprueba en qué fila se encuentra el flujo del algoritmo, de forma que, si se encuentra en la primera fila de las imágenes (v=1), la única forma de reutilización posible es la *reutilización parcial* por columnas hasta haber completado dicha fila y pasar a la siguiente, donde ya se podrán reutilizar también los resultados parciales de la primera fila. Por lo tanto, si el algoritmo se halla iterando la primera fila (*v=1*), el flujo correspondería al mostrado en el apartado *(b)* de la figura, el cual coincide con el diagrama de flujo de la *figura 2.1.1.3* para la *reutilización parcial* por columnas, mientras que, una vez superada la primera fila, a partir de ahí se podrán reutilizar los resultados de las filas anteriores ya procesadas, por tanto si el flujo se sitúa en cualquier otra fila de las imágenes que no sea la primera, el diagrama continúa con el mostrado en el apartado *(c)* de la figura, donde se opera de la misma manera que en el del apartado *(b)*, solo que además en este caso se reutilizan todos los resultados parciales posibles tanto de filas como de columnas obtenidos en las iteraciones anteriores.


Etapa_Procesado_ReTot
Inicializa variables
Inicio bucle for filas (índice v) y bucle for columnas (índice u)
NO
v~=1
SI
1
2

(a)


1
SI
u==1
NO
u<=Máxima disparidad + 1
NO
SI
Selecciona los píxeles en el rango [1, u] para comparar
Selecciona los valores apropiados de las ventanas de comparación centradas en dichos píxeles
Calcula y almacena resultados parciales de columnas para su reutilización posterior. Disparidad=0
Realiza suma de las diferencias absolutas (C_{SAD}) mediante reutilización y calcula la disparidad
Fin bucle for filas y fin bucle for columnas
Fin
Selecciona los píxeles en el rango [1, u] para comparar
Selecciona los valores apropiados de las ventanas de comparación centradas en dichos píxeles
Calcula y almacena resultados parciales de columnas para su reutilización posterior
Selecciona los píxeles en el rango [u-Máxima disparidad, u] para comparar
Selecciona los valores apropiados de las ventanas de comparación centradas en dichos píxeles
Calcula y almacena resultados parciales de columnas para su reutilización posterior

(b)



(c)

Figura 2.1.1.4. Flujograma etapa procesado *SAD con reutilización total*.

Hasta aquí las implementaciones para el modelo de *SAD*. A continuación se detallan las implementaciones realizadas para el modelo de *Hamming*.

2.2 Algoritmos basados en el criterio *HAMMING*

Según este modelo, el cálculo de la correspondencia entre los píxeles se lleva a cabo utilizando como función de coste la denominada *distancia de Hamming*, que, entre dos cadenas binarias, corresponde al número de posiciones de bits en los cuales dichas cadenas toman valores diferentes. Para encontrar la correspondencia de un píxel determinado de la imagen izquierda en la imagen derecha, se compone una ventana centrada en dicho píxel, y se contrasta con cada una de las ventanas centradas en los píxeles de la imagen derecha que se desean comparar con él, donde cada uno de los valores de esas ventanas es una cadena binaria, y, para establecer la correspondencia, se asume que la ventana de mayor similitud con respecto a la del píxel que se desea corresponder es aquella que minimiza la *distancia Hamming* entre las cadenas binarias que componen ambas ventanas, quedando finalmente como correspondiente el píxel sobre el que se centra dicha ventana. En esto se basa la implementación de este algoritmo conocida como *Census*.

El algoritmo *Census* consiste, al igual que los algoritmos *Rank* y *Soft-Rank* para el caso de *SAD*, en la aplicación de una transformación sobre el par de imágenes estereoscópicas antes de proceder a realizar el cálculo de la correspondencia. Dicha transformación recibe el nombre de *transformada Census* [13], la cual se realiza para una determinada ventana centrada en cada uno de los píxeles del par estereoscópico. Al realizar la transformada, se sustituye el valor de intensidad del píxel sobre el que se centra la ventana de transformación por una cadena binaria denominada *vector Census*, en la cual queda codificada la información de dicha ventana. El proceso de transformación consiste en comparar el valor de intensidad de cada uno de los píxeles de la ventana de transformación con el píxel central, de tal manera que si el píxel que se está comparando tiene una intensidad menor que la del píxel central se añade un '1' a la cadena, mientras que si la intensidad es mayor que la del píxel central, se añade un '0'. De esta forma, no solamente se almacena el orden de intensidad como ocurre en el algoritmo *Rank* para el caso de *SAD*, sino que además también se almacena la estructura espacial de los píxeles. En la *figura 2.2.1* se presenta un ejemplo de cómo se aplica la *transformada Census* a un píxel mediante una ventana de transformación de dimensiones *5x5*:

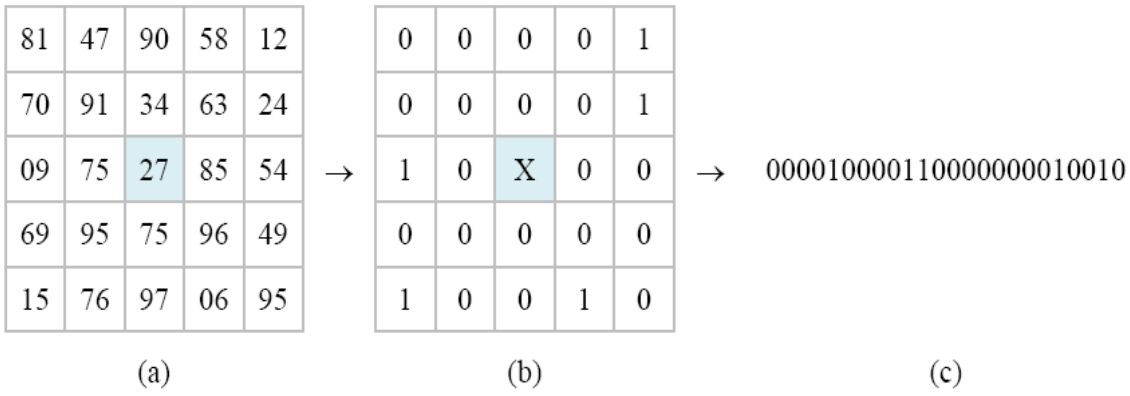


Figura 2.2.1. *Transformada Census* para una ventana de dimensión *5x5*. Ventana sin transformar (a), ventana transformada (b) y *vector Census* (c).

Para este ejemplo, donde el valor central es 27, se tienen 5 píxeles con un valor inferior, los cuales participan con un valor de 1 en su posición correspondiente en la cadena. Nótese que al ser la ventana de tamaño 5x5, ésta tiene 25 elementos, mientras que la cadena tiene 24. Esto se debe a que el píxel central no tiene una posición dedicada dentro del *vector Census*, ya que no se realiza la comparación de dicho píxel consigo mismo. Una vez realizada la transformación de las imágenes, se procede a calcular la correspondencia entre los píxeles mediante las *distancias de Hamming* entre las cadenas que componen las ventanas a comparar, centradas en dichos píxeles.

La etapa de pre-procesado del algoritmo, además de realizar la conversión a escala de grises de las imágenes del par estéreo y añadirles los bordes correspondientes, lleva finalmente a cabo la *transformada Census* sobre ellas. Cabe mencionar que el

algoritmo se ha implementado de forma que la ventana de transformación puede ser de dimensiones distintas a la ventana para el cálculo de la correspondencia. Acto seguido, una vez transformadas las imágenes, se procede con la etapa de procesado, que obtiene el mapa de disparidad mediante la comparación de pares de ventanas de las cadenas binarias que han resultado para cada píxel de las imágenes tras aplicarles la *transformada Census*, empleando el sumatorio de las *distancias de Hamming* entre las cadenas correspondientes en los pares de ventanas como criterio de coste. A continuación se muestra el diagrama de flujo para esta etapa:

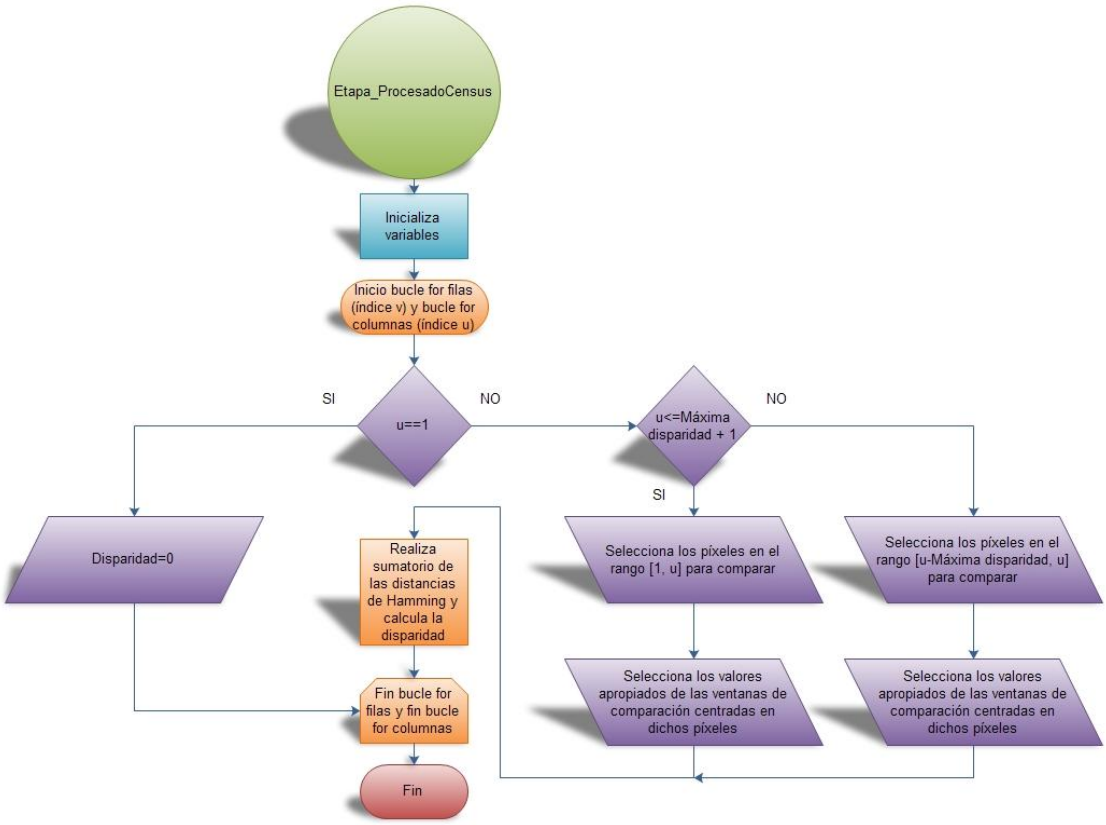


Figura 2.2.2. Flujograma etapa procesado *Census*.

En primer lugar, se preparan las variables necesarias para realizar las operaciones, tras lo cual dan comienzo los bucles anidados que recorrerán las imágenes de arriba hacia abajo (mediante la variable *v*) y de izquierda a derecha (mediante la variable *u*). Siempre que el píxel para el que se desea encontrar su correspondiente sea el primero de cada una de las filas (u=1) de la imagen no será necesario realizar ningún cálculo, ya que en este caso únicamente se compara un solo par de ventanas y por tanto la disparidad siempre resultará nula para los píxeles que ocupan este lugar. En caso de que el flujo del algoritmo se sitúe en cualquier otro píxel de la fila, dependiendo de en qué columna (valor dado por u) se encuentre el píxel se selecciona el rango de píxeles apropiado con los que realizar la comparación (rango [1, u] si u está por debajo del umbral indicado por el valor de máxima disparidad + 1, o bien rango [u – máxima disparidad, u] si u está por encima de dicho umbral) y los valores de las ventanas centradas en dichos píxeles que se deben comparar, y una vez seleccionados y

compuestas las ventanas, se procede con la comparación mediante el sumatorio de las *distancias de Hamming* entre las cadenas correspondientes que componen las ventanas, obteniendo así el mapa de disparidad.

Por último se procede con la etapa de post-procesado en la que se aplica un filtro de mediana con la intención de mejorar la tasa de acierto proporcionada por el algoritmo. Esta etapa se detalla en el subapartado 2.2.1 del presente capítulo.

Existe no obstante una versión más simplificada del algoritmo, cuyo objetivo es reducir significativamente el coste computacional del mismo, a la que se denomina *mini-Census* [20]. En esta versión se realiza la *transformada Census* sobre las imágenes del par estereoscópico antes de calcular la correspondencia de la misma forma que se hacía en la versión completa del algoritmo. La diferencia radica en que, a la hora de calcular la correspondencia, en este caso no se hace entre pares de ventanas de cadenas completas, sino que se realiza únicamente entre pares de cadenas, es decir, en lugar de comparar ventanas completas centradas en el píxel a corresponder, las cuales están compuestas por las cadenas resultantes de la *transformada Census* correspondientes a cada uno de los píxeles que forman dichas ventanas, se comparan únicamente pares de cadenas aisladas, con lo que se reduce el coste computacional del algoritmo.

Los flujogramas correspondientes a la implementación de esta versión reducida del algoritmo son prácticamente idénticos a los ya descritos para la versión completa. La única diferencia es que para la etapa de procesado no es necesario realizar ningún sumatorio de *distancias Hamming*, sino solamente calcular las distancias entre las cadenas de los píxeles correspondientes y comprobar cuál es la menor para establecer la correspondencia puesto que ya no se comparan pares de ventanas de cadenas, sino únicamente pares de cadenas aisladas.

Por otra parte, además de emplear la luminancia de las imágenes para calcular la disparidad, también se puede utilizar información acerca del gradiente de la luminancia. Una forma de hacerlo es mediante el empleo de aproximaciones sencillas del gradiente como el operador *Sobel*. Este operador está basado en los núcleos de convolución S_x y S_y, que sirven para aproximar el gradiente en la dirección horizontal y vertical, respectivamente, los cuales se definen a continuación:

$$S_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

$$S_y = \begin{bmatrix} +1 & +2 & +1 \\ +0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} +1 & 2 & +1 \end{bmatrix}$$

Las aproximaciones del gradiente horizontal G_x y vertical G_y de una imagen de entrada A se obtienen mediante el filtrado con los núcleos de convolución S_x y S_y, tal y como se define a continuación:

$$G_x = S_x * A$$
$$G_y = S_y * A$$

Con el algoritmo *Census* es posible incluir esta información del gradiente aumentando la longitud del *vector Census*, concatenando para formarlo tres vectores que se calculan partiendo de tres imágenes distintas: una de luminancia, otra de la componente horizontal del gradiente y otra de la componente vertical del gradiente, utilizando el operador *Sobel* definido para calcular las componentes. La correspondencia se calcula exactamente de la misma forma que con la versión original del algoritmo *Census*, pero ahora considerando el nuevo vector *Census*, que es el triple de largo que el original. A esta modificación del algoritmo *Census* se la denomina *Census extendido* [13]. No obstante, para el presente proyecto, esta modificación se ha empleado con la versión *mini-Census*, por lo que se la ha denominado *mini-Census extendido*. El hecho de que no se haya aplicado también a la versión original de Census se debe a que la idea de esta versión extendida es, al utilizar el triple de información en las cadenas binarias, aumentar la tasa de acierto, a costa de procesar mayor cantidad de información, lo que se traduce en un mayor tiempo de ejecución. Por tanto, puesto que la versión original de *Census* ya conlleva de por sí un tiempo de ejecución importante (para una aplicación que debe funcionar en tiempo real) se ha considerado más apropiado aplicar esta versión extendida a la implementación *mini-Census*.

La etapa de pre-procesado para esta versión del algoritmo, además de realizar la conversión a escala de grises del par de imágenes estéreo, una vez convertidas las convoluciona con los núcleos de *Sobel* definidos anteriormente obteniendo así las componentes verticales y horizontales del par estéreo, y tras esto, añade los bordes correspondientes a las imágenes y procede a realizar la *transformada Census* obteniendo para cada píxel de cada una de las imágenes del par, tres cadenas binarias diferentes, una correspondiente a la luminancia y las dos cadenas de información extra sobre el gradiente de la luminancia, una para la componente vertical y otra para la componente horizontal. Estas tres cadenas se concatenan una a continuación de la otra, quedando así para cada píxel de cada una de las imágenes del par una cadena binaria cuyo tamaño es el triple que en la versión original del algoritmo. La etapa de procesado es exactamente la misma que para *mini-Census*, solo que se procesan el triple de valores por cada píxel.

Es importante señalar que la principal desventaja del algoritmo *Census* reside en el tamaño del *vector Census*. Nótese que la cantidad de recursos que se requieren para implementar este algoritmo está directamente relacionada con la longitud de la cadena que representa la transformada de cada píxel. Como ya se ha mencionado, el tamaño del *vector Census* es de *((MxM)-1)* bits, siendo *M* el ancho y el alto de la ventana de

transformación. Por tanto, para una ventana de transformación de tamaño *7x7*, por ejemplo, cada uno de los píxeles de las imágenes originales es convertido a un vector de 48 bits. Como resultado, la implementación de este algoritmo demanda una gran cantidad de recursos y conlleva un considerable coste computacional. Recientemente se han hecho progresos a este respecto, reduciendo dicho coste mediante el uso de variaciones del algoritmo tales como la versión de *mini-Census* ya descrita, o la versión denominada *Census disperso*. En esta última versión se reduce el tamaño del *vector Census*, de forma que, en lugar de realizar la transformada para todos los píxeles que componen la ventana de transformación, se selecciona tan solo un subconjunto de esos píxeles, de manera que únicamente se opera sobre el patrón de píxeles seleccionados de la ventana, quedando la cadena binaria resultante para cada píxel de las imágenes reducida en tamaño al número de valores seleccionados que forman el patrón. En la siguiente figura se muestra un ejemplo gráfico para una ventana de tamaño *7x7* y para una ventana de tamaño *5x5*:

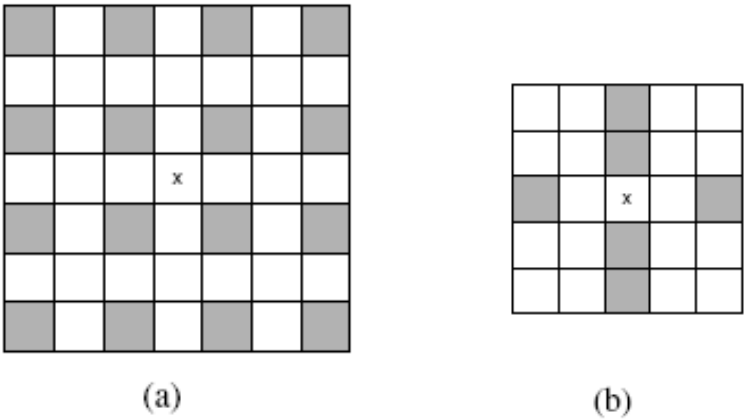


Figura 2.2.3. Patrón de ventana *Census disperso* para una ventana de transformación de *7x7* (a) y para una ventana de transformación de *5x5* (b).

Según el patrón seleccionado para la ventana de *7x7*, la cadena binaria para cada píxel de las imágenes del par estéreo una vez realizada la *transformada Census* quedaría reducida a 16 bits (los correspondientes a la codificación para los píxeles marcados que conforman el patrón disperso) en lugar de los 48 bits que tendría si se emplease la ventana completa, lo cual supone dos terceras partes menos de bits que procesar por cada píxel de las imágenes en el momento de realizar la comparación entre cadenas para calcular las correspondencias. En el caso de la ventana de *5x5* la cadena se reduce a 6 bits, en lugar de los 24 bits que le corresponderían de emplear la totalidad de la ventana, 4 veces menos. Obviamente al utilizar menos bits en las cadenas, la tasa de acierto se reduce con respecto a la que se obtiene utilizando todos los bits, pero, seleccionando un patrón adecuado, se pueden obtener unos resultados muy satisfactorios, de forma que se reduzca considerablemente el tiempo de ejecución, maximizando la tasa de acierto, de modo que la reducción que se produce en la tasa esté dentro de unos márgenes aceptables, viéndose así compensado dicho "sacrificio" por una considerable disminución en el tiempo de ejecución.

Esta versión de *Census* ha sido también implementada para el presente proyecto, y su diagrama de flujo es idéntico al diagrama de flujo de la versión original del algoritmo, salvo para la etapa de pre-procesado, en la que se utiliza una máscara consistente en una matriz cuyo tamaño es el de la ventana de transformación que se desee aplicar a las imágenes, y cuyos valores se establecen a 0, en caso de que dicha posición en la ventana no se quiera operar, o bien a 1, en el caso de que la posición correspondiente de la ventana se desee incluir en la cadena. Dicha máscara por tanto contiene el patrón disperso que se desee aplicar, de forma que sólo se opera sobre los píxeles de la ventana cuyo valor de la máscara está activo (valor 1) y no sobre los que están a 0 en la máscara. Esta misma idea se ha empleado para realizar otra implementación, pero que, en lugar de aplicarla sobre la transformación, se aplica sobre la etapa de procesado, de tal forma que la *transformada Census* se realiza de forma normal, aplicándose en este caso la máscara de patrón disperso sobre las ventanas de coste, de tal manera que en el proceso de obtención del mapa de disparidad, en lugar de comparar todas las cadenas de valores que componen las ventanas de coste, tan solo se comparan las de las posiciones que están marcadas a 1 en la máscara.

Por otra parte, se ha implementado también, al igual que se hizo con *SAD*, una versión con reutilización del algoritmo, ya que esta idea de reutilización de valores que se explicó en el subapartado 2.1.1 de *SAD* para reducir el coste computacional también es aplicable al algoritmo de *Census*, para el cual, en el presente proyecto, se ha implementado su versión con *reutilización parcial* por columnas. El diagrama de flujo correspondiente a la etapa de procesado de este algoritmo (ya que la etapa de pre-procesado se mantiene invariante con respecto a la versión original ya descrita en este apartado), es idéntico al mostrado en la *figura 2.1.1.3* para el algoritmo *SAD con reutilización parcial* por columnas, solo que en lugar de realizar el cálculo de la correspondencia mediante la suma de diferencias absolutas, se lleva a cabo mediante la suma de las *distancias de Hamming*.

2.2.1 Etapa de post-procesado: filtro de mediana

Por último, para finalizar con este segundo capítulo, se presenta la implementación que se ha realizado de un *filtro de mediana*, el cual se introduce como una etapa de post-procesado (etapa que se aplica una vez obtenido el mapa de disparidad) que se puede emplear con cualquiera de los algoritmos basados en la función de coste de las *distancias Hamming* que han sido presentados en este apartado. Los algoritmos de correspondencia basados en la *transformada Census* presentan una cantidad considerable de resultados erróneos que aparecen en el mapa de disparidad en forma de ruido granular. Para reducir este efecto, se ha implementado un *filtro de mediana* que se aplica al mapa de disparidad una vez obtenido. A continuación se muestra en la siguiente figura el diagrama de flujo correspondiente a esta etapa de post-procesado:

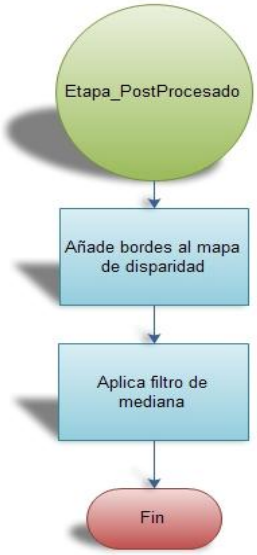
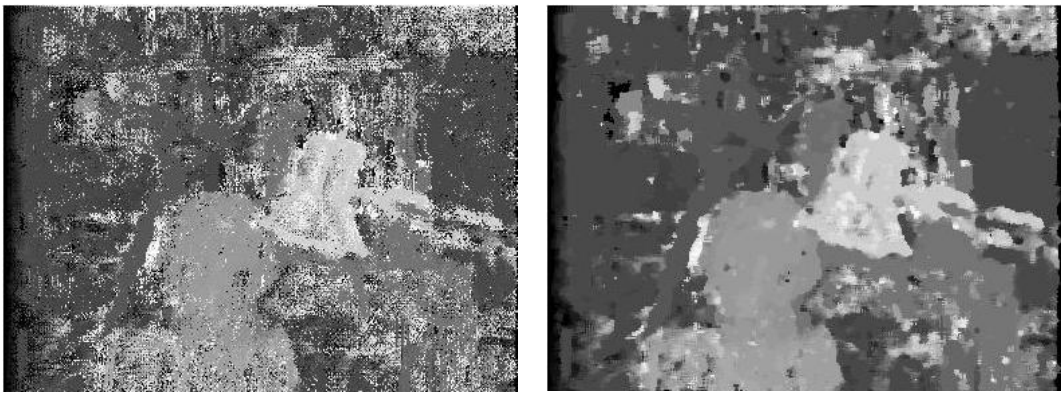


Figura 2.2.1.1. Flujograma etapa de post-procesado.

Este tipo de filtro sustituye cada píxel de una imagen por el valor de la mediana de los píxeles situados en una ventana centrada en dicho píxel. A continuación se muestra una imagen donde se aprecia un ejemplo de un mapa de disparidad antes y después de aplicarle el filtro de la mediana:



(a) (b)

Figura 2.2.1.2. Mapa de disparidad antes de aplicar el filtro de mediana (a) y después de aplicar el filtro (b).

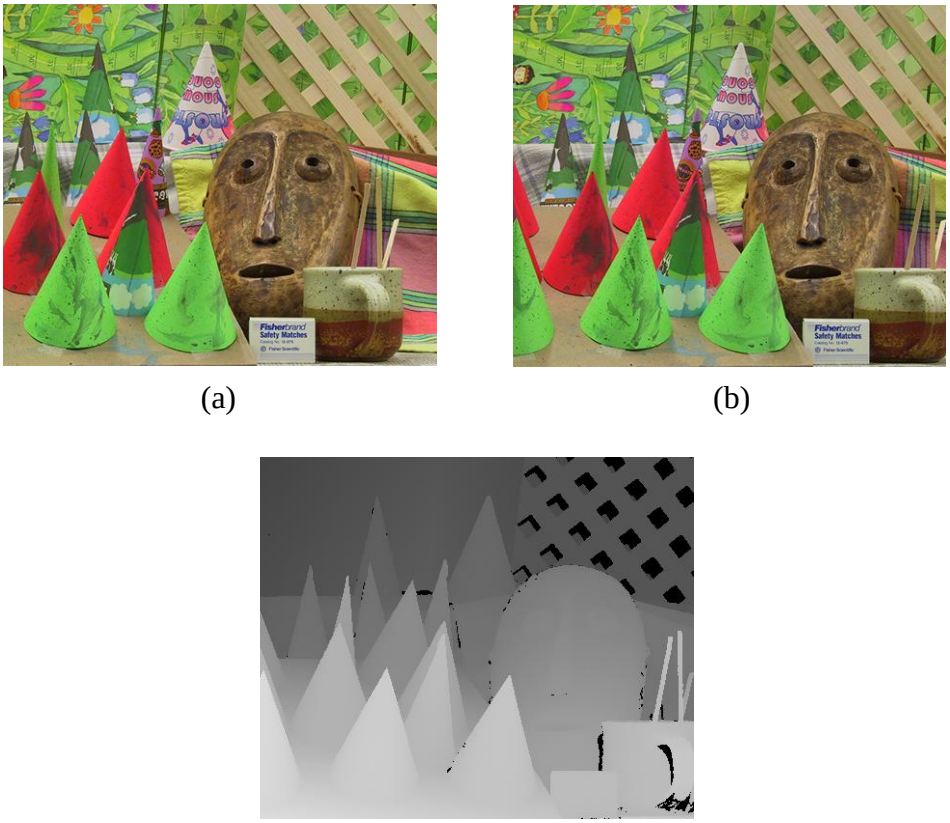
Escogiendo tamaños de ventana pequeños, los *filtros de mediana* eliminan el ruido en gran medida y los bordes apenas se ven alterados, mejorando de esta manera la tasa de acierto para el mapa de disparidad obtenido tras la aplicación del algoritmo sobre las imágenes estereoscópicas de la escena.

Los resultados obtenidos para la implementación de todos los algoritmos mencionados en este capítulo, se exponen a continuación, en el capítulo 3.

CAPÍTULO 3. RESULTADOS OBTENIDOS PARA LAS IMÁGENES ESTÁNDAR

Para evaluar la calidad funcional de los algoritmos implementados descritos en el capítulo anterior, se han llevado a cabo una serie de pruebas en las cuales se ha procedido a la aplicación de dichos algoritmos sobre una determinada selección de imágenes.

En primer lugar, las imágenes utilizadas han sido las pertenecientes a la *Universidad de Middlebury*, las cuales están específicamente diseñadas y constituyen un estándar para la validación y el diseño de algoritmos de visión estereoscópica. Posteriormente, en el capítulo 4, se mostrará su misma eficacia sobre una selección de imágenes reales específicas tomadas en uno de los laboratorios de la *Universidad Politécnica de Cartagena* mediante un sistema de *webcam* que se detallará también en el capítulo 4. A continuación, las *figuras 3.1, 3.2, 3.3* y *3.4* muestran las imágenes de *Middlebury* que se han utilizado para el análisis de los algoritmos.



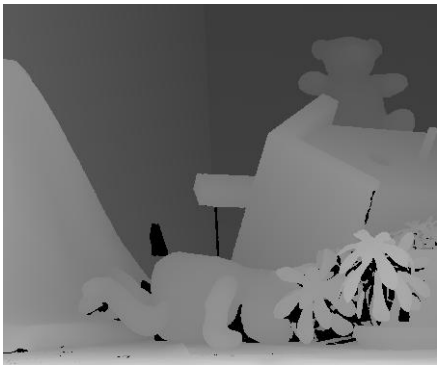
(a) (b)

(c)

Figura 3.1. Imagen *Cones* de *Middlebury*. Izquierda (a), derecha (b) y mapa de disparidad ideal relativo a la imagen derecha (c).

(a)

(b)

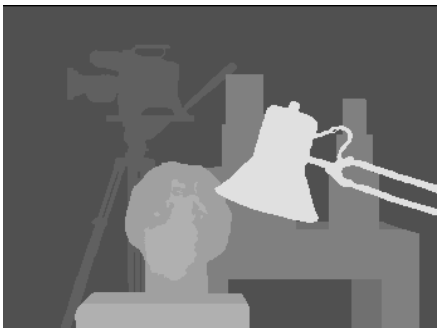


(c)

Figura 3.2. Imagen *Teddy* de *Middlebury*. Izquierda (a), derecha (b) y mapa de disparidad ideal relativo a la imagen derecha (c).

(a)

(b)

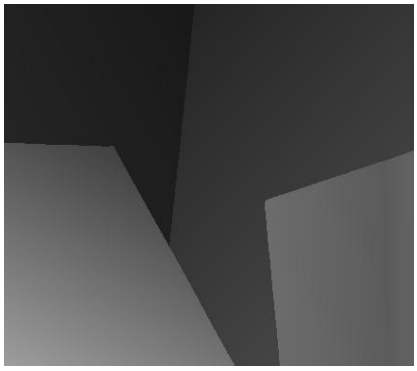


(c)

Figura 3.3. Imagen *Tsukuba* de *Middlebury*. Izquierda (a), derecha (b) y mapa de disparidad ideal relativo a la imagen derecha (c).

(a)

(b)



(c)

Figura 3.4. Imagen *Venus* de *Middlebury*. Izquierda (a), derecha (b) y mapa de disparidad ideal relativo a la imagen derecha (c).

Las imágenes *Cones* y *Teddy* son las más grandes con un tamaño de *450x375* píxeles (*anchoxalto*), mientras que el de *Venus* es algo menor con *434x383* píxeles, siendo *Tsukuba* la imagen más pequeña de las cuatro con un tamaño de *384x288* píxeles. Esta diferencia de tamaños permitirá observar en los resultados la variación del

comportamiento de los algoritmos ante la variación en la cantidad total de píxeles a procesar. A continuación en la *tabla 3.1* se muestra el valor de los parámetros de *máxima disparidad* (d_{max}) y *umbral* que se han establecido para cada imagen a la hora de realizar las pruebas:

	Cones	***Teddy***	***Tsukuba***	***Venus***
d_{max}	51	46	35	16
Umbral	4	6	7	3

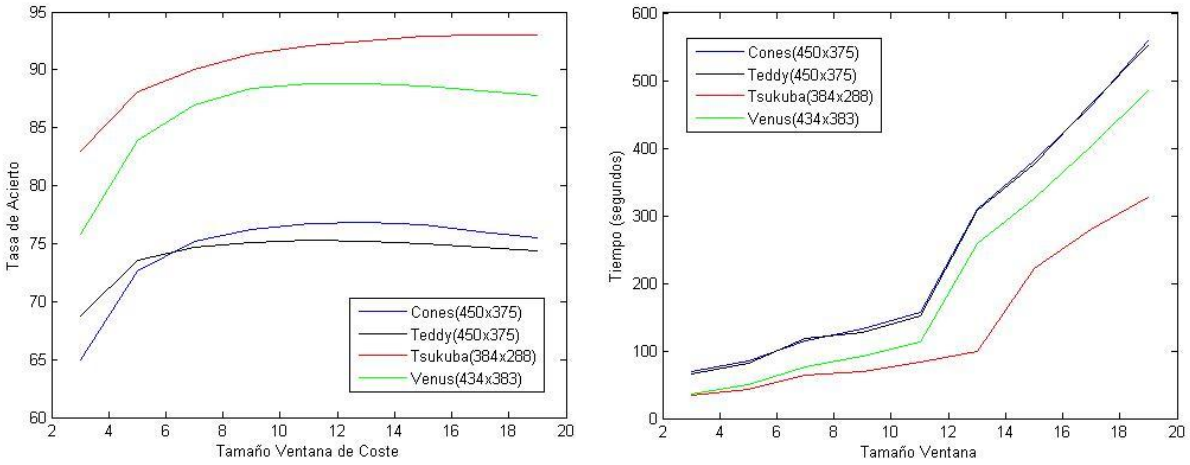
Tabla 3.1. Parámetros d_{max} y *umbral* para las imágenes de *Middlebury*.

El parámetro *umbral* se emplea a la hora de calcular la tasa de acierto de los algoritmos, que se obtiene realizando la diferencia entre los valores del mapa de disparidad resultante y los del mapa de disparidad ideal, e indica la diferencia máxima que se tolera entre dichos valores para cada una de las imágenes. Si la diferencia supera este umbral establecido, el valor que se esté contrastando se considerará como erróneo, en caso contrario, se considerará correcto. Una vez conocida la cantidad de píxeles del mapa que están correctos, se obtiene el porcentaje de acierto dividiendo esta cantidad entre el número total de píxeles del mapa y multiplicando por cien.

A continuación se muestran los resultados obtenidos en dos apartados, uno para los algoritmos basados en el criterio *SAD* y otro para los algoritmos basados en el criterio de *Hamming*.

3.1 Resultados obtenidos para los algoritmos basados en *SAD*

La *figura 3.1.1* muestra gráficamente para cada una de las imágenes de forma conjunta la evolución de la tasa de acierto así como del tiempo de ejecución con respecto al tamaño de ventana de coste para el algoritmo de *SAD* básico (sin haber realizado ninguna transformación previa sobre las imágenes originales). Se ha variado la ventana de coste desde un tamaño de *3x3* hasta un tamaño de *19x19*.



(a) (b)

Figura 3.1.1. Gráficas tasa de acierto (a) y tiempo de ejecución (b) para *SAD* básico frente al tamaño de la ventana de coste.

Tal y como puede apreciarse en la gráfica (a) de la figura anterior, la tasa de acierto aumenta conforme aumenta el tamaño de la ventana de coste, produciéndose un punto óptimo para un cierto tamaño de ventana, a partir del cual pasado ese tamaño la tasa de acierto comienza a disminuir de nuevo. No obstante, salvo para los primeros tamaños de ventana, las diferencias no son demasiado marcadas, lo cual se puede observar en la siguiente tabla que muestra los valores exactos del porcentaje de acierto para cada tamaño e imagen:

	Tamaño Ventana de Coste								
	3x3	***5x5***	***7x7***	***9x9***	***11x11***	***13x13***	***15x15***	***17x17***	***19x19***
Cones	64.97%	72.61%	75.14%	76.20%	76.75%	**76.86%**	76.61%	76.06%	75.46%
Teddy	68.78%	73.57%	74.63%	75.13%	**75.28%**	75.20%	74.99%	74.66%	74.32%
Tsukuba	83.01%	88.08%	90.07%	91.37%	92.11%	92.51%	92.89%	93.02%	**93.04%**
Venus	75.86%	83.88%	86.98%	88.40%	88.78%	**88.80%**	88.56%	88.21%	87.81%

Tabla 3.1.1. Tasa de acierto respecto al tamaño de la ventana de coste para el algoritmo *SAD* básico.

Se ha resaltado para cada imagen el valor de mayor porcentaje de acierto conseguido. En todos los casos se supera el 75% de acierto, y en algunos (como es el caso de *Venus y Tsukuba*) el 85% y el 90%. La mayor diferencia se produce al pasar de la ventana de *3x3* a la de *5x5*, no superando esta diferencia en ningún caso el 8%, y en el rango de ventanas de *9x9* a *19x19* la variación que se produce no supera el 1% salvo para el caso de *Tsukuba* donde aun así permanece inferior al 2%. Este dato es interesante si se observan los tiempos de ejecución, representados en la gráfica (b) de la

figura 3.1.1, puesto que el tiempo de ejecución del algoritmo sí que se ve sensiblemente afectado conforme aumenta el tamaño de la ventana de coste, sobretodo precisamente en el rango de ventanas de *9x9* a *19x19*, en el que apenas varía el porcentaje de acierto. Observando la *tabla 3.1.2* se puede apreciar con mayor precisión:

	Tamaño Ventana de Coste								
	3x3	***5x5***	***7x7***	***9x9***	***11x11***	***13x13***	***15x15***	***17x17***	***19x19***
Cones	69.80	85.86	115.42	132.77	156.46	309.82	382.38	461.98	559.52
Teddy	65.36	81.30	117.80	127.98	151.01	308.42	375.55	465.39	551.43
Tsukuba	34.87	43.26	63.88	69.08	83.49	99.14	221.42	279.68	327.05
Venus	36.44	50.30	77.12	91.81	113.27	258.08	325.08	402.07	484.23

Tabla 3.1.2. Tiempo de ejecución (expresado en segundos) respecto al tamaño de la ventana de coste para el algoritmo *SAD* básico.

Como cabía esperar, el tiempo de ejecución para *Cones* y *Teddy* es prácticamente el mismo ya que ambas tienen el mismo tamaño, y para la imagen de *Tsukuba* el tiempo de ejecución es notablemente menor que para el resto, ya que es la imagen de menor tamaño con una marcada diferencia con respecto a las otras tres. Como ya se ha comentado anteriormente, se aprecia que el tiempo de ejecución es mucho más sensible a la variación del tamaño de la ventana de coste que el porcentaje de acierto. En el caso de *Cones* y *Teddy*, al pasar de un tamaño de ventana de *11x11* a *13x13*, el tiempo de ejecución prácticamente se ve duplicado, y en el caso de *Tsukuba* y de *Venus* al pasar de un tamaño de ventana de *13x13* a *15x15* y de *11x11* a *13x13*, respectivamente, el tiempo pasa a ser más del doble. Esto podría ser un indicio de que es más conveniente utilizar tamaños de ventana de coste reducidos, ya que el porcentaje de acierto (que marca la calidad del mapa de disparidad obtenido) apenas se ve afectado, y se reduce considerablemente el tiempo de ejecución del algoritmo.

Para los algoritmos *Rank* y *Soft-Rank*, además del tamaño de la ventana de coste también entra en juego el tamaño de la ventana de transformación, que no tiene por qué coincidir con el tamaño de la ventana de coste, por lo que, para el análisis de estos dos algoritmos, además de variar la ventana de coste desde un tamaño de *3x3* a *15x15* se han empleado ventanas de transformación de *3x3*, *5x5* y *7x7* para cada una de esas ventanas de coste, para así poder observar cómo varía el comportamiento de los algoritmos con la variación de ambas ventanas de forma conjunta. A continuación, la *figura 3.1.2* muestra gráficamente para el algoritmo *Rank* la evolución de dicho comportamiento en lo que al porcentaje de acierto se refiere para cada una de las imágenes de *Middlebury*:

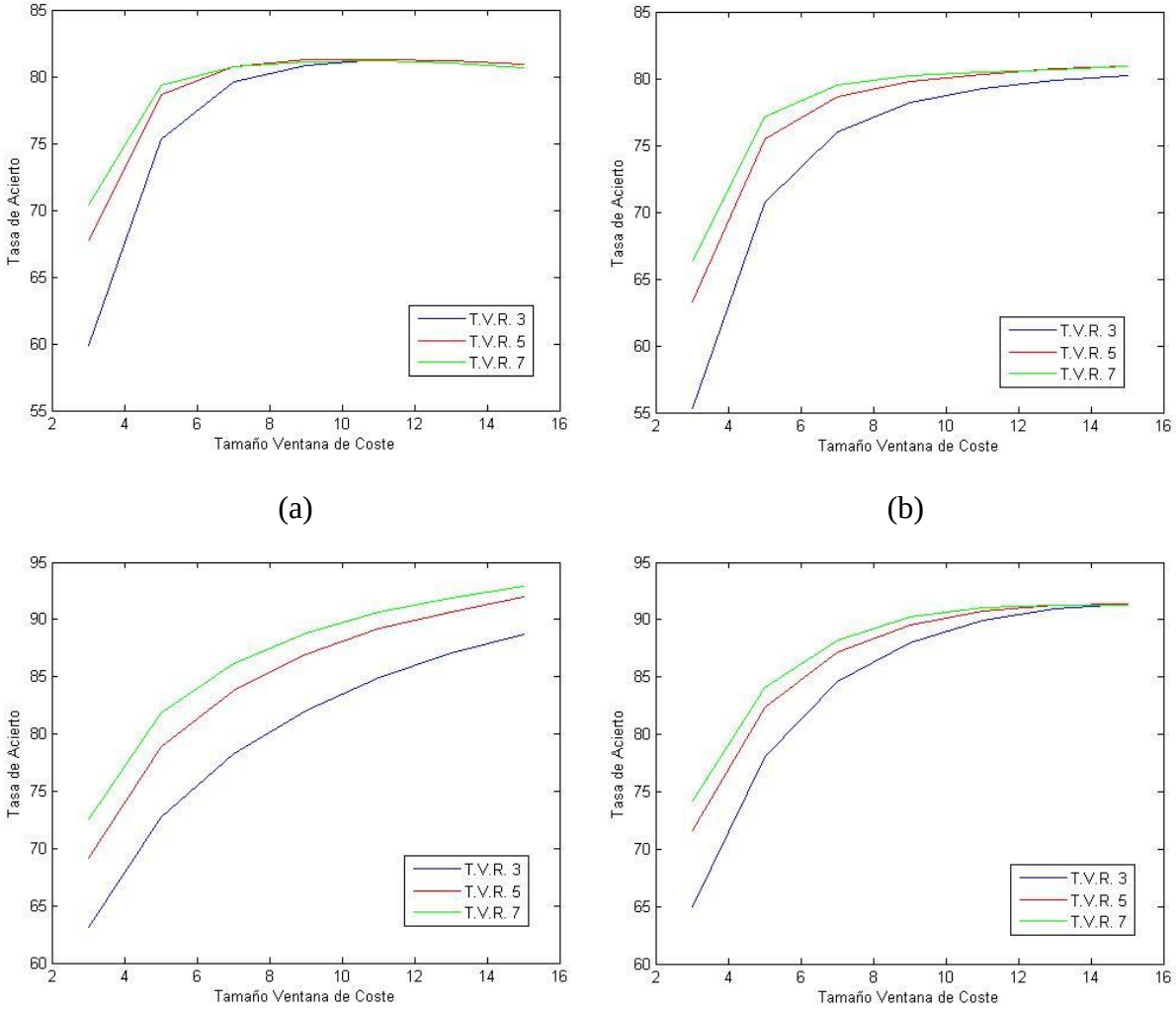


Figura 3.1.2. Gráficas tasa de acierto para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y la ventana de transformación para el algoritmo *Rank*.

En todas las gráficas se muestra de forma superpuesta para la misma imagen la evolución de la tasa de acierto con respecto al tamaño de la ventana de coste para los tamaños de ventana de transformada *Rank* de *3x3* (*T.V.R. 3*, en color azul), *5x5* (*T.V.R. 5*, en color rojo) y *7x7* (*T.V.R. 7*, en color verde), tal y como se indica en la leyenda. Observando los resultados se puede afirmar que la variación de la ventana de transformación afecta sobre todo en los tamaños de ventana de coste reducidos, donde a mayor tamaño de ventana de transformación mejor tasa de acierto se obtiene, pero conforme se va aumentando el tamaño de la ventana de coste, las tasas de acierto que se producen terminan por converger a un mismo valor con lo que el tamaño de ventana de transformación empleado va perdiendo importancia conforme va creciendo el de la ventana de coste. Este dato puede ser interesante y se comprobará al observar los tiempos de ejecución. En cuanto a la evolución que se produce en la tasa de acierto con

respecto a la variación de la ventana de coste, se obtienen las mismas conclusiones que se han observado y comentado para el algoritmo básico de *SAD*. A mayor ventana de coste, mejor tasa de acierto sobre todo para tamaños reducidos, hasta un cierto tamaño a partir del cual la tasa se estabiliza en una zona más o menos lineal con muy poca variación. La *figura 3.1.3* muestra gráficamente la evolución de los tiempos de ejecución (expresados en segundos) para cada una de las imágenes de *Middlebury* en función de la ventana de coste y la ventana de transformación empleadas:

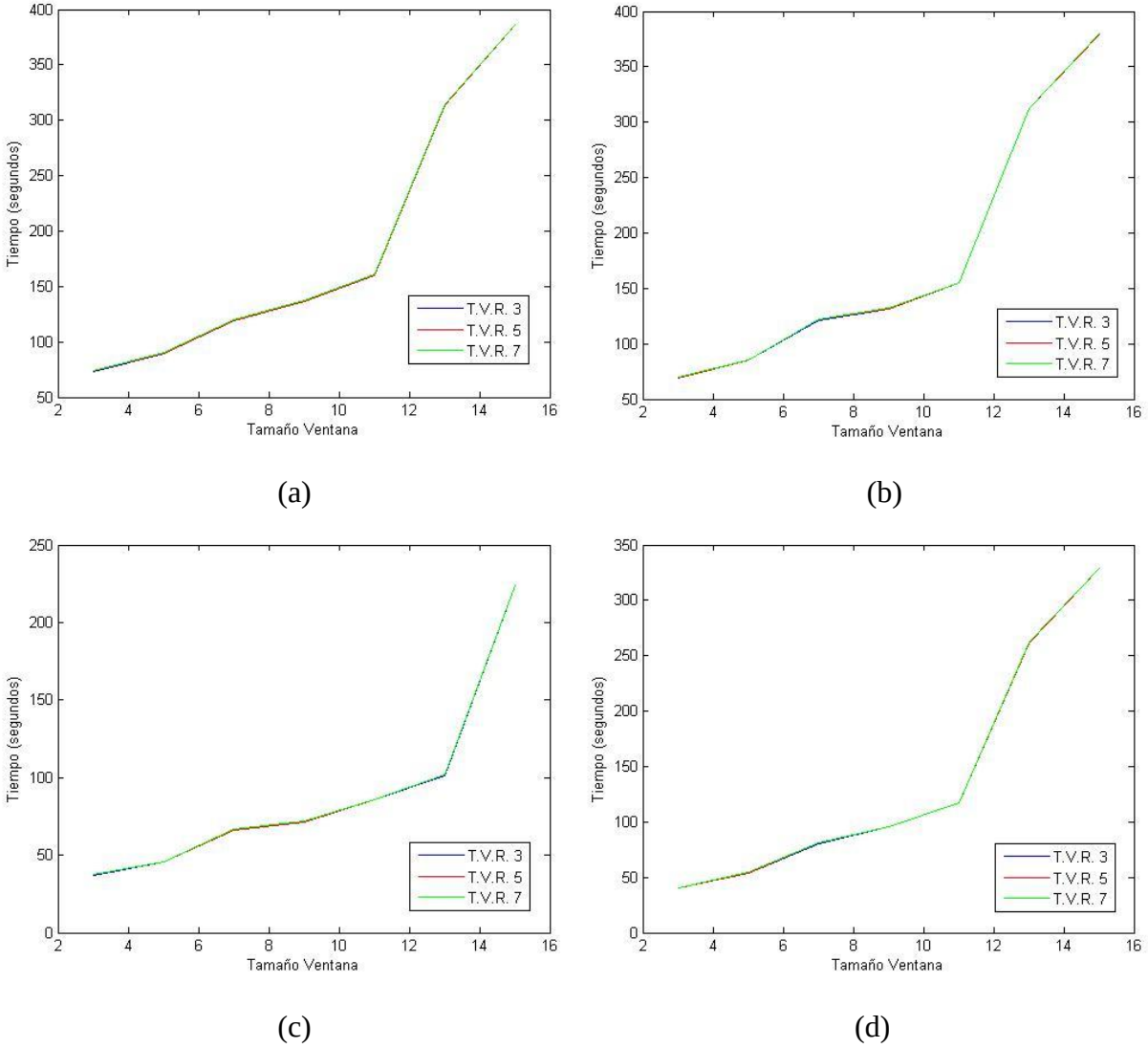


Figura 3.1.3. Gráficas tiempo de ejecución para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y la ventana de transformación para el algoritmo *Rank*.

Se puede afirmar que el tiempo de ejecución no se ve afectado (apenas existen unas diferencias de milésimas de segundo) prácticamente por la variación en el tamaño de la ventana de transformada *Rank*, por eso en todas las gráficas los tres colores (azul, rojo y verde) que representan la evolución temporal para cada uno de los tres tamaños de ventana de transformación diferentes aparecen prácticamente uno encima de otro

dando la sensación de que solamente hay una línea pintada en las gráficas. Por este motivo, es interesante el dato que se ha mencionado anteriormente de que la tasa de acierto se vea mejorada con una mayor ventana de transformación para los tamaños de ventana reducidos. Puede ser un indicativo de que conviene utilizar tamaños de ventana de transformación grandes, con tamaños de ventana de coste reducidos, ya que podría mejorar la tasa de acierto sin afectar prácticamente al tiempo de ejecución del algoritmo. No obstante, la *figura 3.1.4* muestra las gráficas correspondientes a la comparativa de las tasas de acierto obtenidas por el algoritmo *SAD* y el algoritmo *Rank* para cada una de las cuatro imágenes:

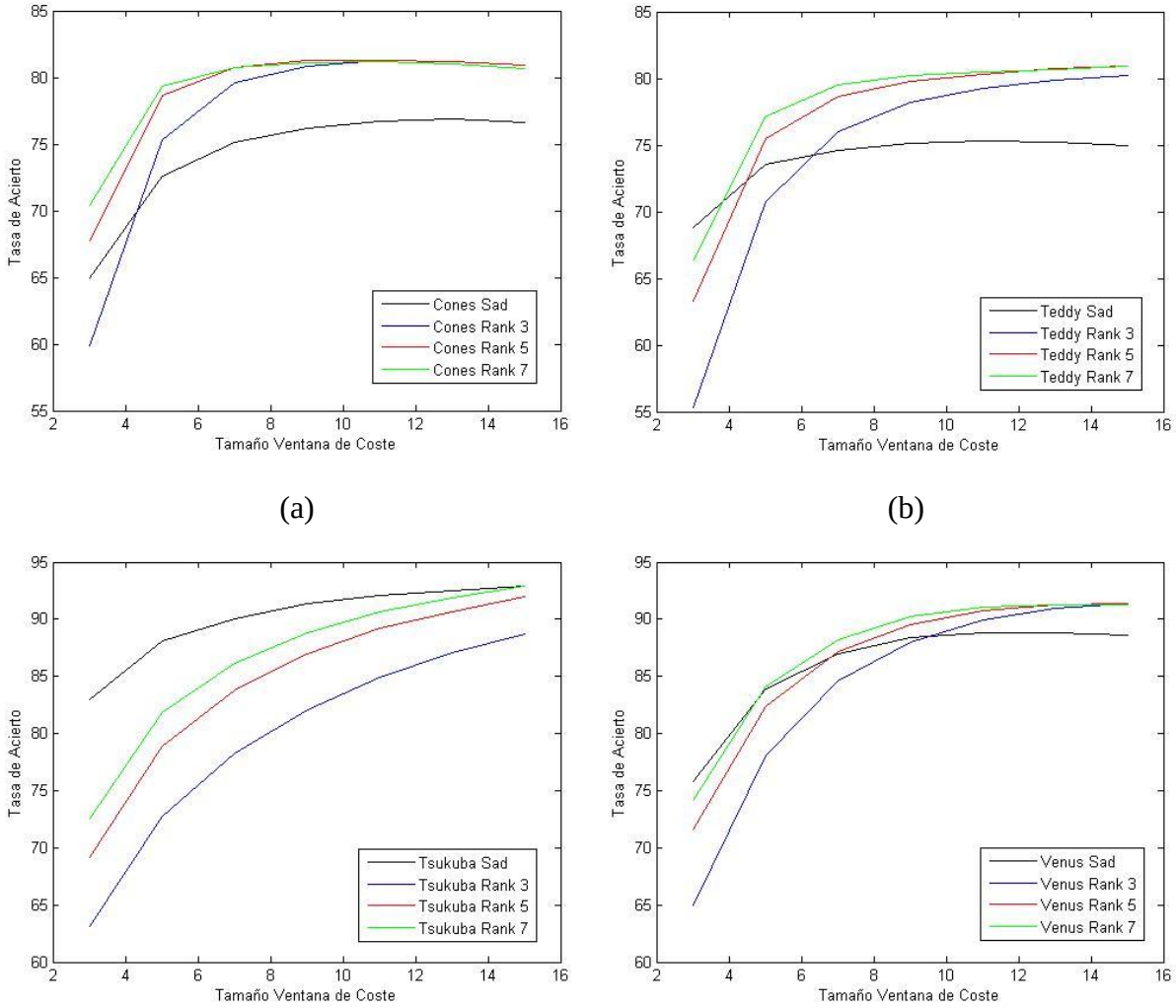


Figura 3.1.4. Gráficas comparativas de la tasa de acierto para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y la ventana de transformación para los algoritmos *SAD* y *Rank*.

En dichas gráficas se representa de forma conjunta para cada imagen en colores azul, rojo y verde respectivamente las tasas de acierto conseguidas por el algoritmo *Rank* para las ventanas de transformación de *3x3*, *5x5* y *7x7*, y en color negro la tasa de

acierto conseguida por el algoritmo de *SAD* básico, tal y como se indica en la leyenda. Se puede observar que unas imágenes (como es el caso de *Cones* y *Teddy*) responden mejor ante el algoritmo *Rank* que otras (como *Venus* y *Tsukuba*), pero la tendencia en todos es la misma y es que conforme se va aumentando la ventana de coste, el algoritmo *Rank* produce una cierta mejora en la tasa de acierto con respecto al algoritmo *SAD*. En el caso de *Cones* y *Teddy* esta mejora se produce ya desde los tamaños de ventana de coste más reducidos, mientras que en el caso de *Venus* y *Tsukuba* esta mejora con respecto a *SAD* se ve "retrasada" hasta emplear tamaños de ventana de coste mayores y además es de menor amplitud, sobre todo para el caso de *Tsukuba* en el que tan solo en los últimos puntos representados en la gráfica empieza a verse como las líneas de evolución de la tasa de acierto para *Rank* comienzan a sobrepasar a las de *SAD* básico y que es siempre en todas las pruebas el caso en el que la tendencia del comportamiento de los algoritmos se "estira" más en el eje de las *x*, es decir, el comportamiento es el mismo sobre *Tsukuba* que sobre las demás imágenes, pero éste se produce de forma más progresiva. Sin embargo, a pesar de que se produce mejoría, ésta no es demasiado marcada, lo cual se puede apreciar con mayor precisión en la siguiente tabla:

	Tamaño Ventana de Coste						
	3x3	***5x5***	***7x7***	***9x9***	***11x11***	***13x13***	***15x15***
SAD	64.97%	72.61%	75.14%	76.20%	76.75%	**76.86%**	76.61%
Rank 3	59.83%	75.34%	79.58%	80.87%	**81.25%**	81.18%	80.96%
Rank 5	67.79%	78.68%	80.74%	81.24%	**81.31%**	81.18%	80.92%
Rank 7	70.37%	79.34%	80.71%	81.07%	**81.14%**	80.96%	80.68%

Tabla 3.1.3. Tasas de acierto respecto al tamaño de la ventana de coste para los algoritmos *SAD* básico y *Rank* sobre la imagen *Cones*.

La *tabla 3.1.3* muestra los valores exactos representados en la gráfica (a) de la *figura 3.1.4*, referente a las tasas de acierto sobre la imagen de *Cones* para los algoritmos de *SAD* básico y de *Rank* (para las 3 ventanas de transformación diferentes que se han aplicado). En las gráficas se puede apreciar como la mejora del algoritmo *Rank* con respecto a *SAD* básico es mayor para la imagen de *Cones* que para el resto, es por ello que se ha elegido mostrar en la tabla los valores exactos para esta imagen, y se han resaltado los valores máximos de tasa de acierto que se han obtenido con cada algoritmo. De esta manera, se puede ver que efectivamente el algoritmo *Rank* consigue mejores resultados que *SAD*, en este caso particular por dos razones, una es que el máximo porcentaje de acierto se alcanza con un tamaño de ventana de coste menor, y otra es que además dicho máximo es mayor que el alcanzado mediante el algoritmo *SAD* básico. Sin embargo, cabe destacar que esta diferencia entre el mejor porcentaje conseguido por *SAD* y el conseguido por *Rank* es inferior al 5%, lo cual no representa una gran ventaja y como se ha comentado y se puede apreciar en las gráficas en los casos de *Teddy*, *Tsukuba* y *Venus* esta diferencia es aún menor. Es por tanto interesante observar el comportamiento de los tiempos de ejecución, el cual se muestra a continuación en la *figura 3.1.5*:

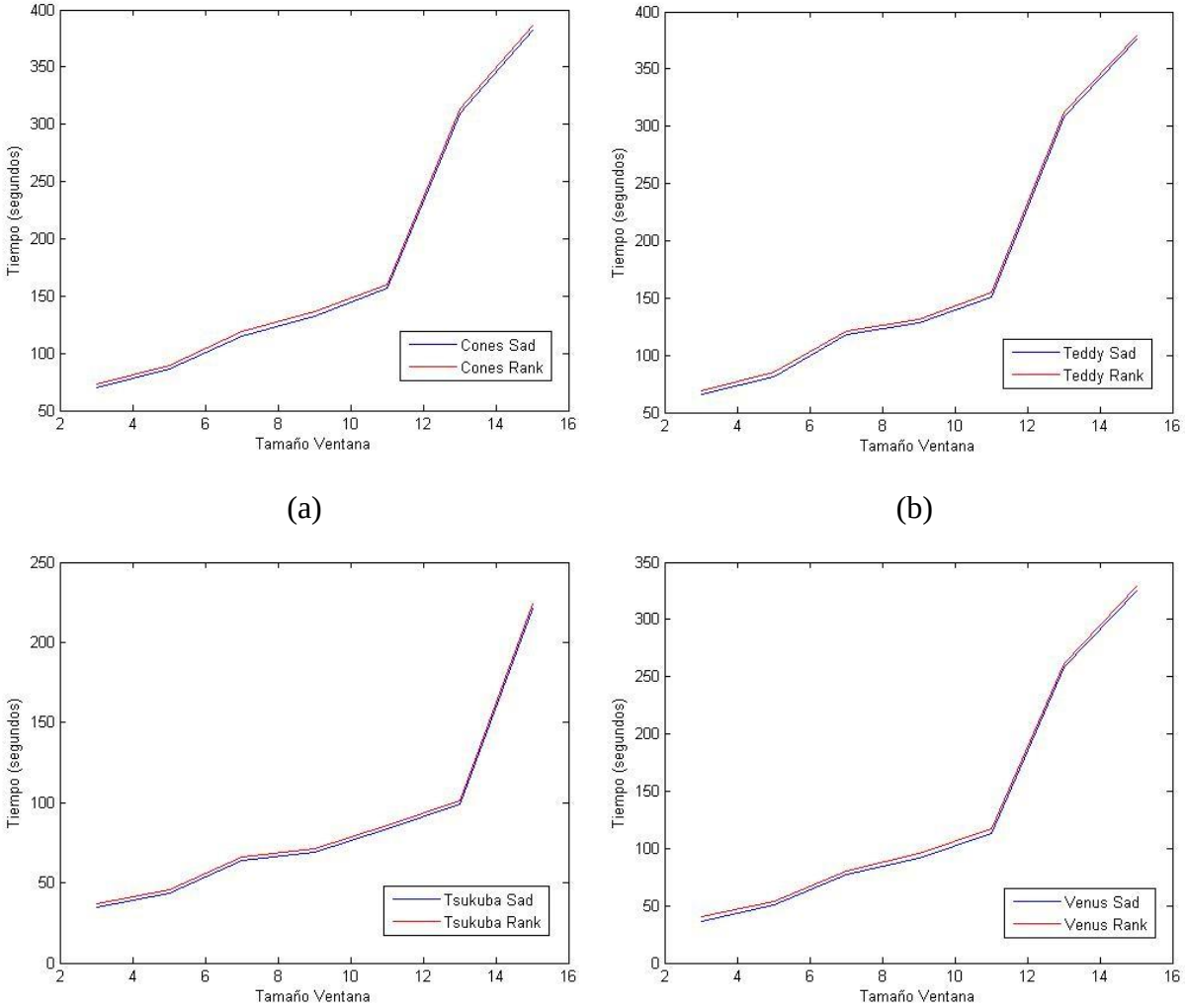


Figura 3.1.5. Gráficas comparativas del tiempo de ejecución para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste para los algoritmos *SAD* y *Rank*.

Como cabía esperar, las curvas que representan la evolución de los tiempos de ejecución de sendos algoritmos sobre cada imagen son idénticas entre sí, algo muy lógico teniendo en cuenta que *Rank* emplea el mismo criterio y por lo tanto el mismo proceso para el cálculo de la disparidad que emplea el algoritmo *SAD* básico. La clave radica en que para el caso de *Rank* hay que añadir el tiempo que se invierte en realizar el proceso de transformación sobre las imágenes originales previo al cálculo de la disparidad mediante el criterio *SAD*, y ésa es la pequeña diferencia en el tiempo de ejecución de sendos algoritmos que se aprecia en las gráficas. Esta diferencia, según para qué imagen y ventana de transformación utilizada, varía entre 2.3 segundos (diferencia mínima, producida para *Tsukuba* con una ventana de transformación de *3x3*) y 4.2 segundos (diferencia máxima, producida para *Cones* y *Teddy* con una ventana de transformación de *7x7*). De manera que, teniendo en cuenta estos datos y los analizados

anteriormente, utilizar *Rank* en lugar de *SAD* básico puede reportar una mejora en la tasa de acierto inferior a un 5% (en el mejor de los casos) a costa de tardar de 2 a 4 segundos más (en el mejor de los casos). No se puede afirmar por tanto de forma clara según estos datos que sea mejor utilizar *Rank* que *SAD*. Será necesario estudiar el comportamiento de los algoritmos sobre las imágenes reales específicas, lo cual se aborda en el capítulo 5 del presente informe.

Para el caso del algoritmo *Soft-Rank* se puede llegar a unas conclusiones bastante similares como se verá a continuación. La *figura 3.1.6* muestra gráficamente para el algoritmo *Soft-Rank* de forma conjunta la evolución del porcentaje de acierto para cada una de las imágenes de *Middlebury* frente a una variación del tamaño de la ventana de coste desde *3x3* a *15x15* para los tres tamaños diferentes (*3x3*, *5x5* y *7x7*) de ventana de transformación *Soft-Rank* que se han aplicado:

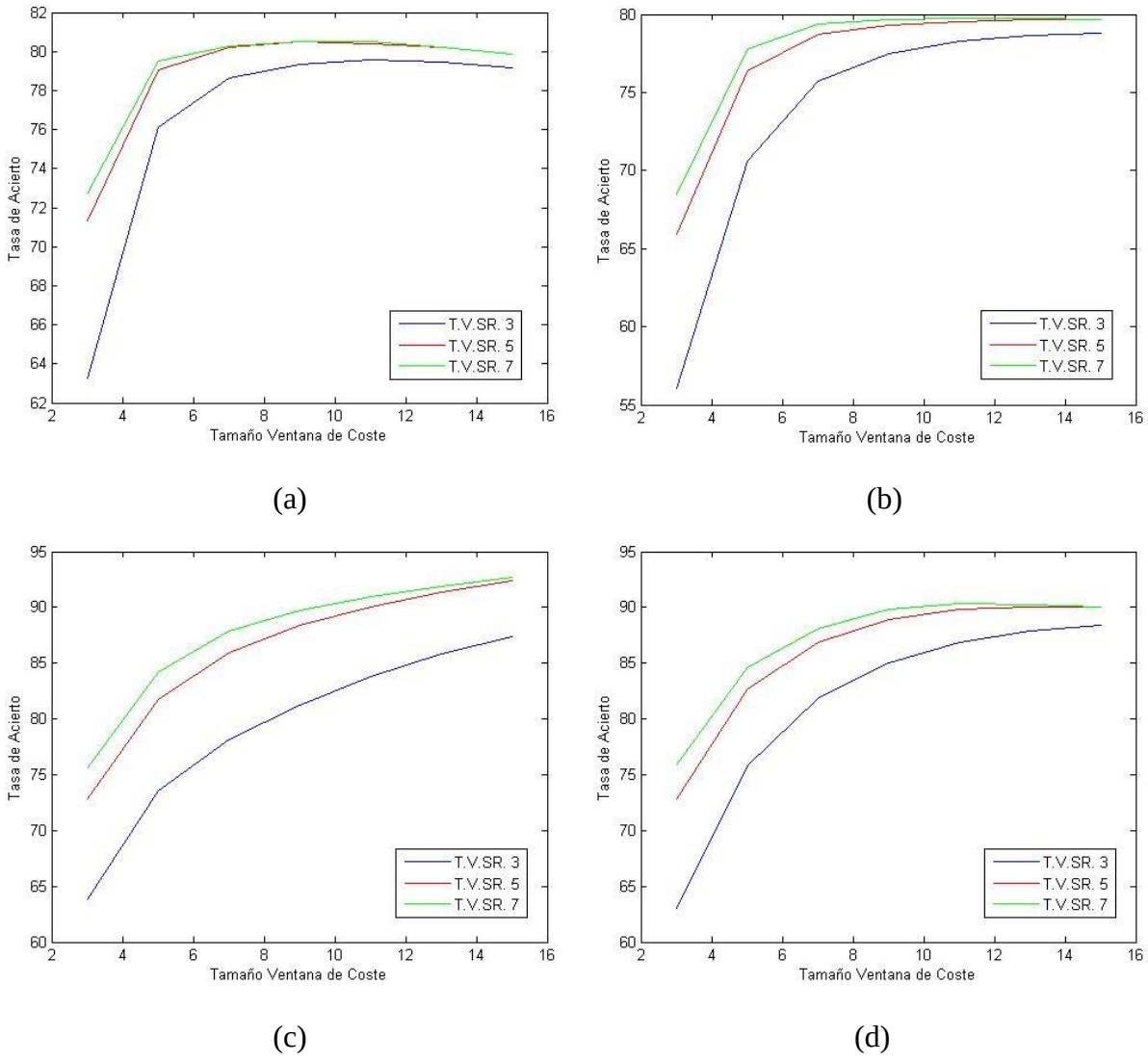


Figura 3.1.6. Gráficas tasa de acierto para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y la ventana de transformación para el algoritmo *Soft-Rank*.

La tendencia en el comportamiento de la tasa de acierto que puede observarse con respecto a la variación en el tamaño de la ventana de transformación y de la ventana de coste es prácticamente la misma que se ha observado para el algoritmo *Rank* anteriormente. Se aprecia una diferencia algo más marcada entre el empleo del tamaño de la ventana de transformación más reducido (*3x3*) y los tamaños superiores, no obstante es una diferencia mínima y por lo demás el comportamiento es el mismo, aumentando el tamaño de la ventana de coste los diferentes tamaños de la ventana de transformación tienden a converger hacia un mismo valor en la tasa de acierto y a mayor ventana de coste mejor porcentaje de acierto hasta estabilizarse en un rango más o menos lineal a partir de un determinado tamaño. En la *figura 3.1.7* pueden observarse los tiempos de ejecución en función de la ventana de coste y la ventana de transformación empleada:

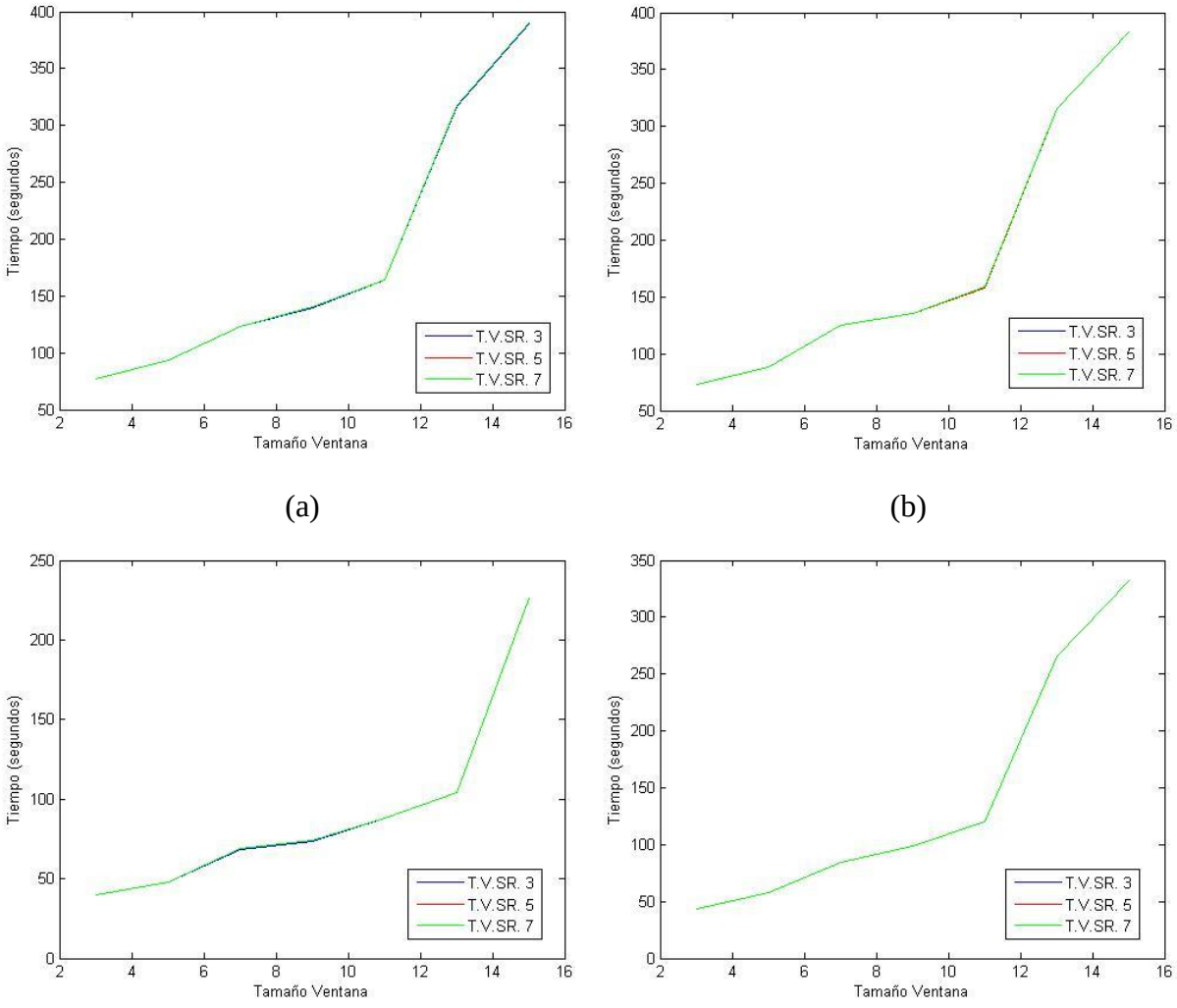


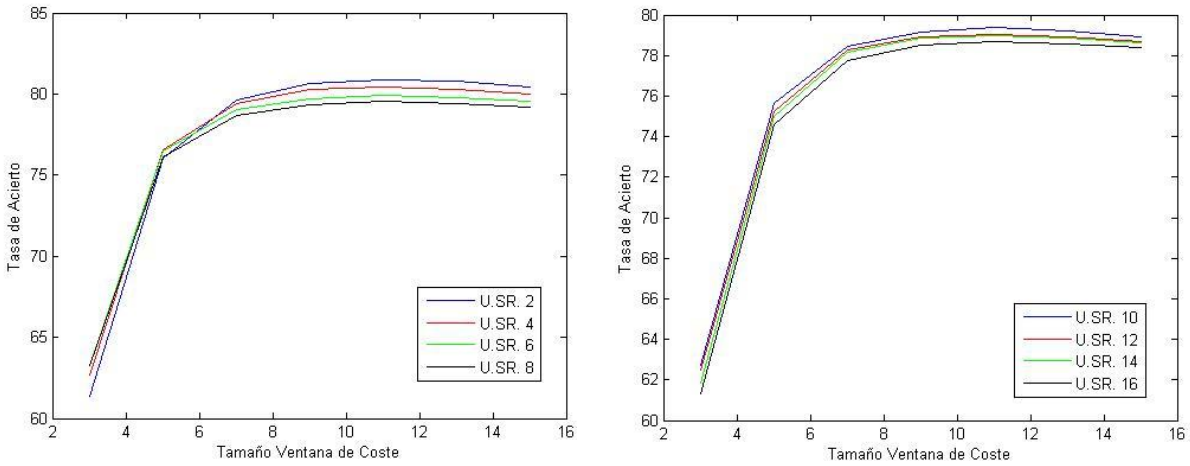
Figura 3.1.7. Gráficas tiempo de ejecución para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y la ventana de transformación para el algoritmo *Soft-Rank*.

De nuevo, al igual que en el caso de *Rank*, se puede afirmar con rotundidad que la variación en el tamaño de la ventana de transformación no afecta prácticamente (apenas en milésimas de segundo en el peor de los casos) al tiempo de ejecución del algoritmo, con lo cual no hay ningún problema en emplear tamaños de ventana de transformación grandes en tanto en cuanto sea conveniente para obtener una mejora en la tasa de acierto.

Para el algoritmo *Soft-Rank* se ha querido comprobar además la influencia que tiene la variación del parámetro *t* que aparece en la ecuación que define la transformada *Soft-Rank*, al que se ha denominado como *umbral Soft-Rank*. Dicha función (que ya fue mostrada en el capítulo anterior) era la siguiente:

$$I_{SoftRank}(\boldsymbol{p}) = \sum_{q \in N_p} \min\left(1, \max\left(0, \frac{I(\boldsymbol{p}) - I(\boldsymbol{q})}{2t} + \frac{1}{2}\right)\right)$$

Se ha llevado a cabo por tanto una variación de dicho parámetro desde un valor de 2 a 16 para observar su influencia sobre la tasa de acierto (ya que sobre el tiempo de ejecución como es lógico no tiene ningún efecto debido a que su cambio no altera de ningún modo el proceso de ejecución empleado). Los resultados gráficos para esta comprobación se muestran en la *figura 3.1.8*:



(a) (b)

Figura 3.1.8. Gráficas tasa de acierto con respecto al *Umbral Soft-Rank* sobre la imagen *Cones* para una variación del parámetro de 2 a 8 (a) y de 10 a 16 (b).

Se ha dividido el resultado en dos gráficas separadas para que el amontonamiento de las líneas en una misma imagen gráfica no dificulte la apreciación visual del resultado. Puede observarse fácilmente de esta manera que la tasa de acierto empeora al aumentar el valor del umbral, si bien es cierto que la diferencia que se produce es muy pequeña, siendo ésta algo más marcada entre los valores más reducidos

del parámetro que entre los valores más grandes. A continuación mediante la *tabla 3.1.4* puede apreciarse esta pequeña diferencia con mayor precisión:

	Tamaño Ventana de Coste						
	3x3	***5x5***	***7x7***	***9x9***	***11x11***	***13x13***	***15x15***
U.SR. 2	61.34%	76.05%	79.64%	80.63%	**80.87%**	80.77%	80.45%
U.SR. 4	62.69%	76.52%	79.43%	80.24%	**80.43%**	80.27%	80.00%
U.SR. 6	63.26%	76.48%	79.04%	79.73%	**79.91%**	79.80%	79.58%
U.SR. 8	63.24%	76.11%	78.64%	79.33%	**79.57%**	79.44%	79.17%
U.SR. 10	62.73%	75.62%	78.45%	79.13%	**79.35%**	79.21%	78.91%
U.SR. 12	62.46%	75.22%	78.24%	78.89%	**79.01%**	78.89%	78.67%
U.SR. 14	61.85%	74.98%	78.14%	78.86%	**78.98%**	78.87%	78.62%
U.SR. 16	61.31%	74.56%	77.75%	78.52%	**78.70%**	78.58%	78.37%

Tabla 3.1.4. Tasas de acierto respecto al *Umbral Soft-Rank* empleado sobre la imagen *Cones*.

Se ha resaltado en cada caso la mejor tasa de acierto obtenida por el algoritmo, pudiéndose apreciar con facilidad el hecho comentado anteriormente de que a mayor valor del *umbral Soft-Rank* la tasa de acierto empeora. Sin embargo, analizando los datos mostrados en las gráficas y en la tabla, cabe mencionar que esta diferencia entre usar un valor de umbral u otro es realmente pequeña. La mayor diferencia se da al pasar de usar un valor de 2 a un valor de 16 y ésta es de un 2.17%. Este mismo comportamiento se produce para el resto de imágenes de *Middlebury* de forma menos marcada aún y por ello sólo se han mostrado los datos exactos para *Cones* a modo de ejemplo, por lo cual ante estos datos se puede afirmar que prácticamente es irrisorio el valor de umbral que se emplee al aplicar el algoritmo *Soft-Rank*, siendo en todo caso más beneficioso emplear un valor pequeño que uno grande.

Si se realiza la comparación del algoritmo *Soft-Rank* con los algoritmos *SAD* y *Rank*, se pueden alcanzar unas conclusiones muy similares a las que se observaron en la comparativa de *Rank* con respecto a *SAD*. La *figura 3.1.9* muestra esta comparativa (en lo que a la tasa de acierto se refiere) de cada uno de estos tres algoritmos para las imágenes de *Middlebury*:

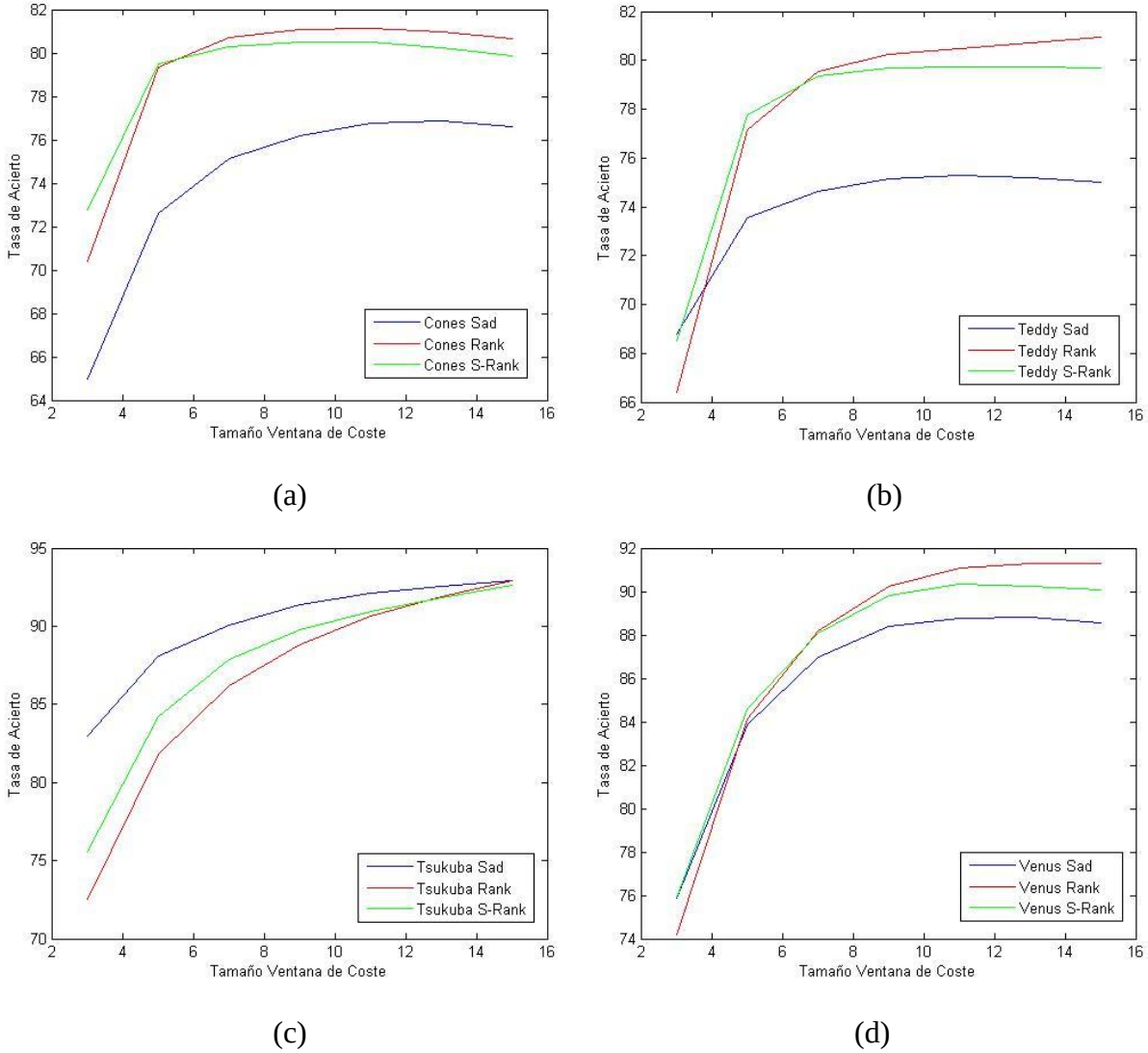


Figura 3.1.9. Gráficas comparativas de la tasa de acierto para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste para los algoritmos *SAD*, *Rank* y *Soft-Rank*.

Se puede apreciar en los resultados que el algoritmo *Soft-Rank* produce una cierta mejora en la tasa de acierto con respecto a *SAD* (al igual que ocurría para *Rank*), pero en términos generales no mejora con respecto al propio *Rank* (salvo para los tamaños de ventana de coste más reducidos). En cuanto al tiempo de ejecución, es lógico esperar que el de *Soft-Rank* sea el mayor de los tres algoritmos, ya que el proceso de transformación de las imágenes originales conlleva mayor complejidad que el de *Rank* y por lo tanto requiere de un mayor coste computacional lo que se traduce en un mayor tiempo de ejecución del algoritmo. Se puede comprobar que efectivamente se cumple esta deducción lógica observando los resultados gráficos mostrados en la *figura 3.1.10*:

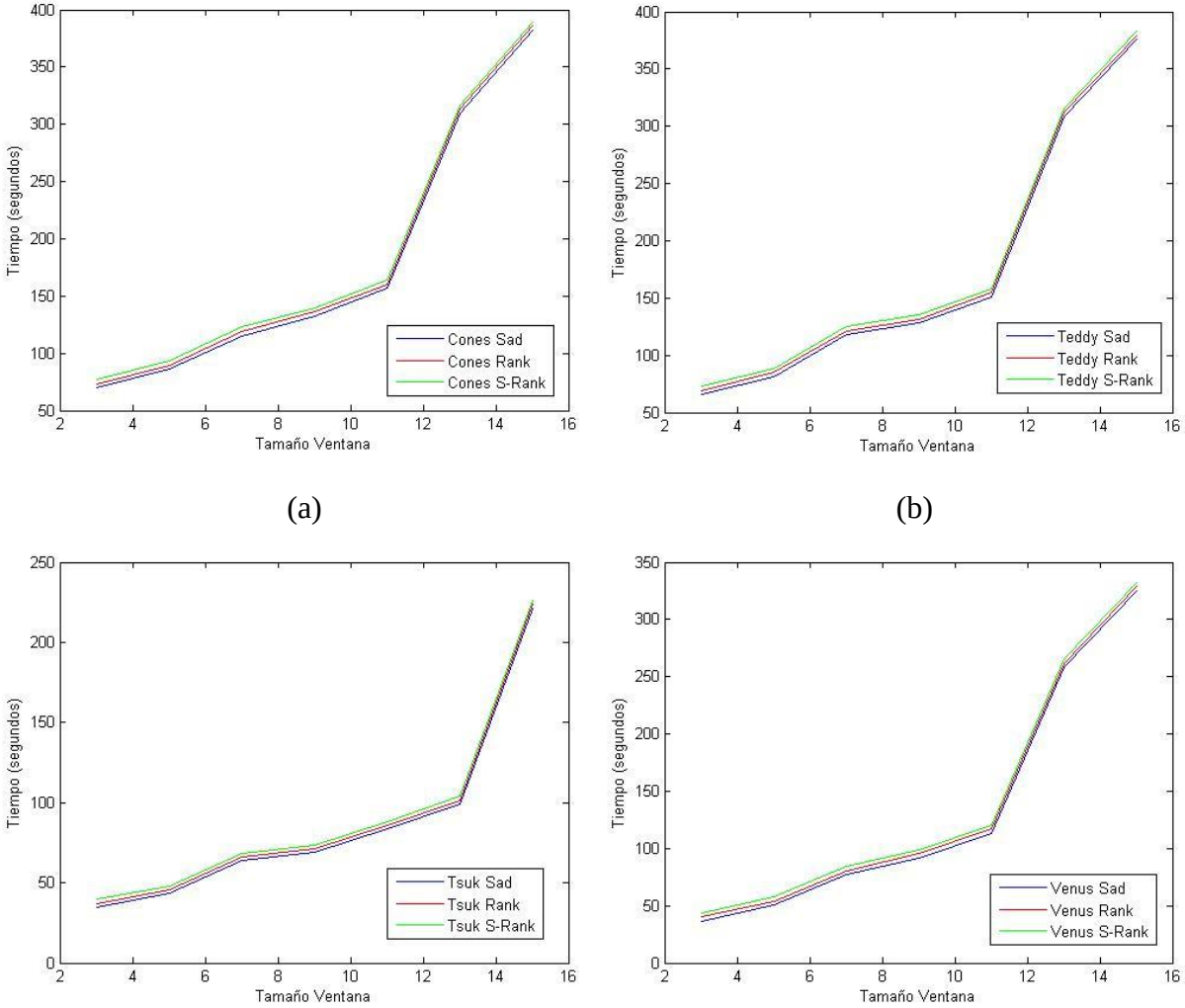


Figura 3.1.10. Gráficas comparativas del tiempo de ejecución para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste para los algoritmos *SAD*, *Rank* y *Soft-Rank*.

La *figura 3.1.10* muestra gráficamente la comparación de los tiempos de ejecución entre los tres algoritmos para cada una de las imágenes de *Middlebury* y es fácil observar que *Soft-Rank* es el que más tiempo "se cobra" en su ejecución tal y como cabía esperar. La diferencia entre *Rank* y *SAD* era de entre 2.3 segundos (en el mejor de los casos) y 4.2 segundos (en el peor de los casos), mientras que la que se produce entre *Soft-Rank* y *SAD* es de entre 4.7 segundos (en el mejor de los casos) y 7.2 segundos (en el peor de los casos).

A la vista de estos resultados se puede concluir respecto al algoritmo *Soft-Rank* que mejora la tasa de acierto producida por *SAD*, pero esta mejora es menor que la que proporciona el algoritmo *Rank*, que ya de por sí era bastante pequeña (inferior a un 5% en el mejor de los casos), a costa de un mayor tiempo de ejecución que el propio

algoritmo *Rank*. De forma y manera que tampoco se puede afirmar que sea más conveniente usar *Soft-Rank* que *SAD* para esta selección de imágenes de *Middlebury* ya que produce una mejora en la tasa de acierto inferior todavía a la que proporciona el empleo de *Rank* conllevando un tiempo de ejecución aún mayor.

Para finalizar con el análisis de la parte correspondiente a los algoritmos que hacen uso del criterio de coste *SAD*, tan solo resta por estudiar los resultados obtenidos para las versiones del algoritmo básico que hacen uso de la técnica de reutilización de valores. Como ya se comentó en el capítulo anterior, esta técnica se ha implementado con la intención de comprobar si se puede lograr una aceleración sustancial en el tiempo de ejecución del algoritmo, habiéndose realizado dos versiones, una con reutilización parcial por columnas y otra con reutilización total, de manera que únicamente hay que observar los resultados obtenidos con respecto al tiempo de ejecución de los algoritmos.

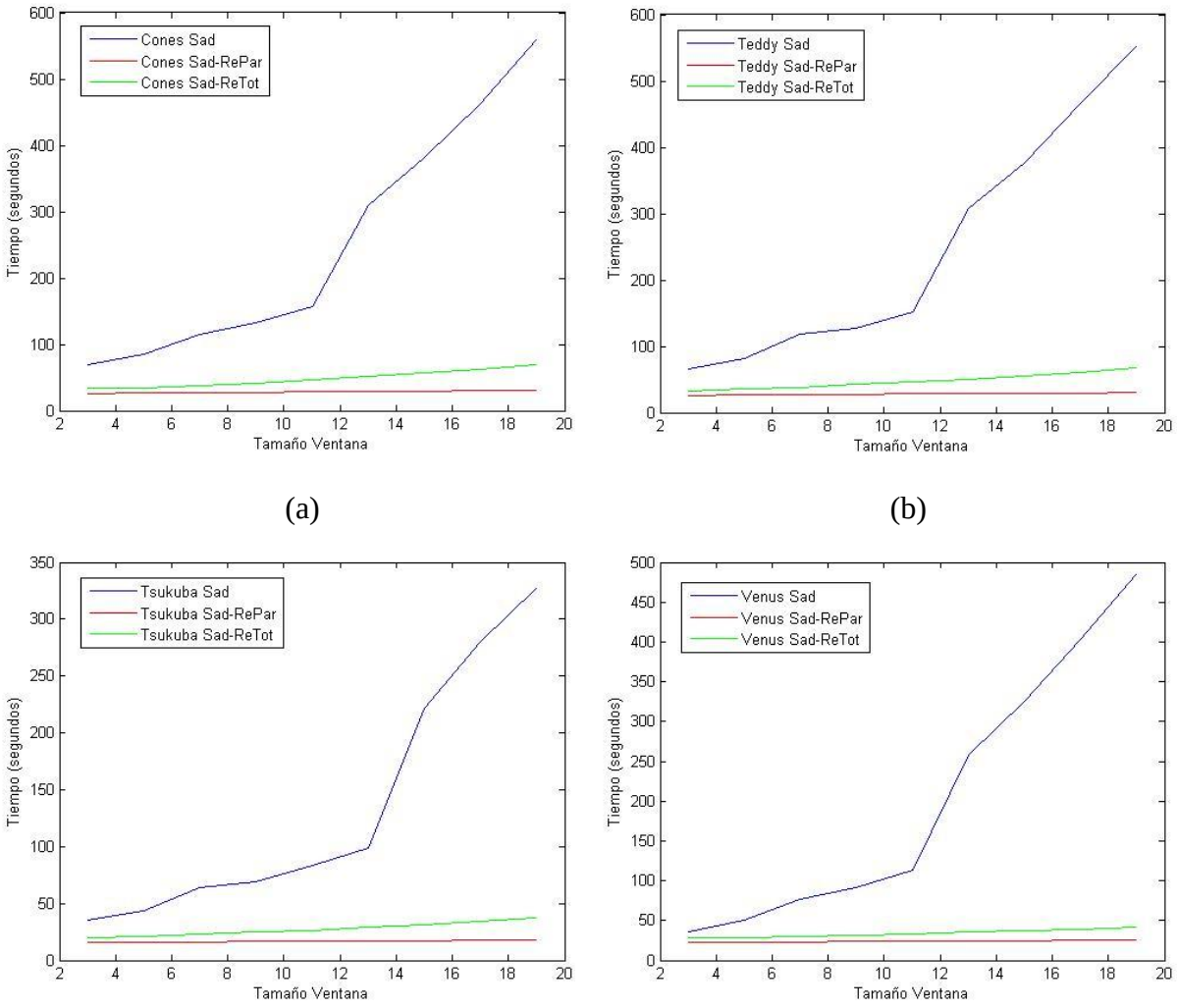


Figura 3.1.11. Gráficas comparativas del tiempo de ejecución para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste para los algoritmos *SAD* y *SAD* con reutilización de valores.

La *figura 3.1.11* muestra gráficamente los resultados para cada una de las imágenes de *Middlebury* obtenidos mediante el algoritmo *SAD* básico con reutilización parcial por columnas y el algoritmo *SAD* básico con reutilización total en comparación con los del algoritmo *SAD* básico original para una variación del tamaño de la ventana de coste de *3x3* a *19x19*. Se puede apreciar con suma claridad que las versiones del algoritmo que hacen uso de la técnica de reutilización proporcionan una mejora ciertamente amplia en el tiempo de ejecución. Concretamente, el que proporciona el mejor de los resultados es la versión del algoritmo con reutilización parcial por columnas, que además se puede observar un dato muy importante y es que el tiempo de ejecución que proporciona es prácticamente lineal y apenas se ve afectado por el aumento del tamaño de la ventana de coste. Este dato es muy importante porque es indicativo de que empleando esta versión del algoritmo se podrían usar los tamaños de ventana de coste más grandes sin que apenas se vea afectado el tiempo de ejecución permitiendo así lograr las mejores tasas de acierto en el mínimo tiempo posible. En la *tabla 3.1.5* se puede ver con mayor exactitud la importancia de esta mejora:

	Tamaño Ventana de Coste								
	3x3	***5x5***	***7x7***	***9x9***	***11x11***	***13x13***	***15x15***	***17x17***	***19x19***
SAD	34.87	43.26	63.88	69.08	83.49	99.14	221.42	279.68	**327.05**
RePar	15.58	15.84	16.08	16.58	16.74	17.05	17.16	17.64	**17.84**

Tabla 3.1.5. Tiempos de ejecución (expresados en segundos) respecto al tamaño de la ventana de coste para los algoritmos *SAD* básico original y *SAD* con reutilización parcial para la imagen *Tsukuba*.

Se ha remarcado el resultado para la ventana de coste con la que se conseguía el mayor porcentaje de acierto para esta imagen, que en este caso era la de mayor tamaño (*19x19*). La versión del algoritmo con reutilización parcial consigue la mejor tasa de acierto 18 veces más rápido que la versión original.

Ante estos resultados se puede finalmente concluir que lo más conveniente para los algoritmos basados en el criterio de coste *SAD* es emplear o bien el algoritmo *SAD* básico original o bien el algoritmo con la transformada *Rank*, empleando para el proceso de cálculo de la disparidad la versión con reutilización parcial de valores. No obstante esto es así para el caso de esta concreta selección de imágenes estándar de la Universidad de *Middlebury*. Será necesario contrastar este comportamiento sobre la selección de imágenes reales específicas en el capítulo 4 del presente proyecto. Se procede a continuación en el subapartado *3.2* del presente capítulo a analizar los resultados obtenidos para los algoritmos basados en el criterio de *Hamming*.

3.2 Resultados obtenidos para los algoritmos basados en *HAMMING*

Para los algoritmos basados en el criterio de *Hamming*, además del tamaño de la ventana de coste también entra en juego el tamaño de la ventana de transformación , que no tiene por qué coincidir con el tamaño de la ventana de coste, por lo que, para el análisis de estos algoritmos, además de variar la ventana de coste también se han empleado ventanas de transformación diferentes para cada una de esas ventanas de coste, pudiendo observar de esta manera cómo varía el comportamiento de los algoritmos con la variación de ambas ventanas de forma conjunta.

A continuación, la *figura 3.2.1* muestra gráficamente para el algoritmo *Census* la evolución de su comportamiento en lo que al porcentaje de acierto se refiere para cada una de las imágenes de *Middlebury*, para una variación de la ventana de coste de *3x3* a *19x19* y ventanas de transformación de *3x3*, *5x5 y 7x7*:

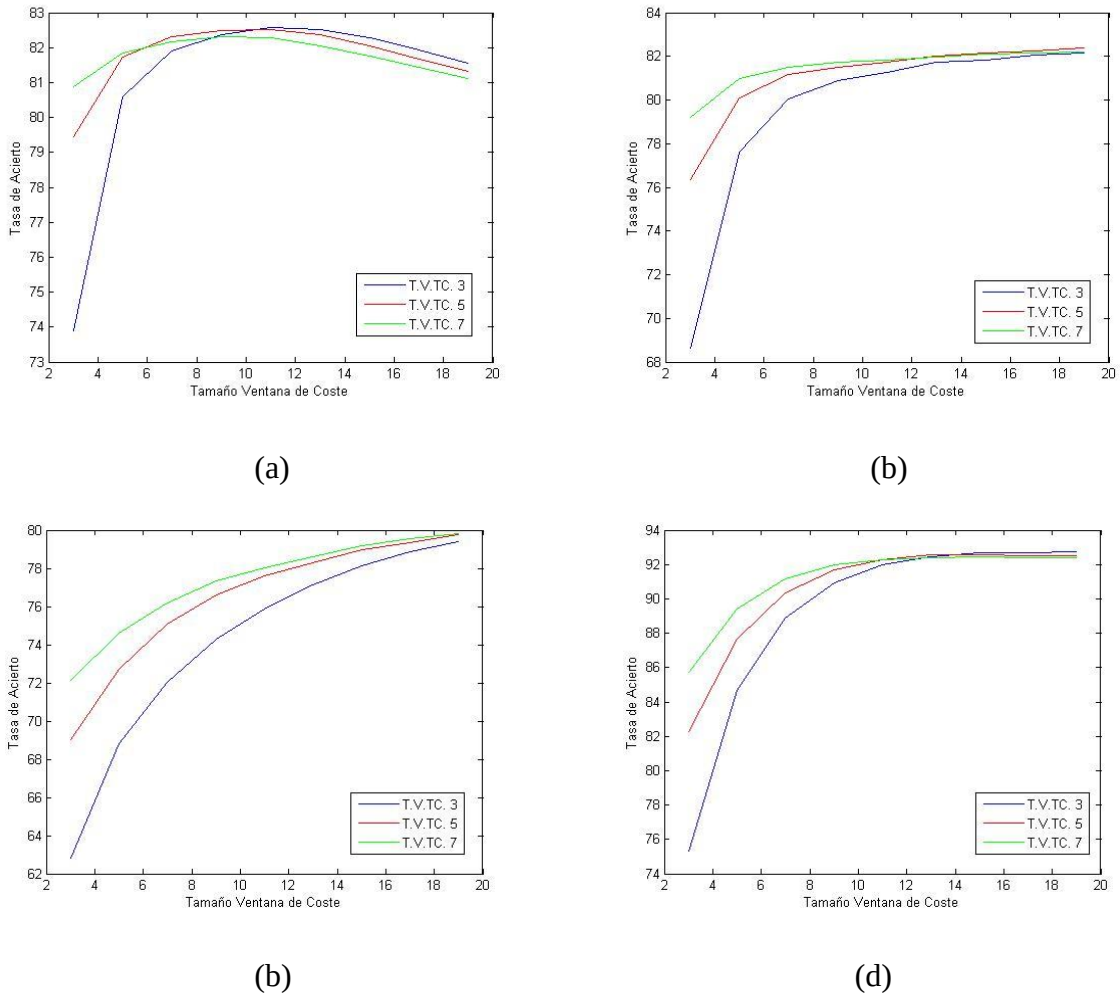


Figura 3.2.1. Gráficas comparativas de la tasa de acierto para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y ventana de transformación para el algoritmo *Census*.

Tal y como puede apreciarse en las gráficas de la figura anterior, el comportamiento es muy similar al de los algoritmos ya mostrados en el apartado de *SAD*. La tasa de acierto aumenta conforme aumenta el tamaño de la ventana de coste, produciéndose un punto óptimo para un cierto tamaño de ventana, a partir del cual pasado ese tamaño la tasa de acierto comienza a disminuir de nuevo. Así mismo, para diferentes tamaños de ventana de transformación, aumentando la ventana de coste todos tienden a converger a un mismo valor, con lo cual, a mayor ventana de coste, el tamaño de la ventana de transformación empleado va careciendo de importancia cada vez en mayor medida, mientras que si se emplean tamaños de ventana de coste reducidos se obtiene mayor tasa de acierto conforme se aumenta el tamaño de la ventana de transformación.

Por otra parte, el tiempo de ejecución requerido por este algoritmo es sumamente elevado, con lo cual se hace interesante contrastar su eficiencia con respecto a la proporcionada mediante el empleo de la técnica de reutilización de valores que ya se aplicó para el algoritmo *SAD* y que también se ha decidido aplicar sobre *Census*, a través de la versión de *Census con reutilización parcial por columnas*. La *figura 3.2.2* muestra el tiempo de ejecución para cada una de las imágenes de *Middlebury*, en función del tamaño de ventana de coste y ventana de transformación empleados, obtenido mediante la técnica de reutilización de valores. Se puede apreciar en ella que el tiempo de ejecución se ve sensiblemente afectado tanto con el aumento de la ventana de coste como con el aumento de la ventana de transformación.

En la *figura 3.2.3* se puede apreciar la diferencia entre el tiempo requerido por el algoritmo *Census* en su versión original y el tiempo demandado por la versión de reutilización de valores para el caso concreto de la imagen *Cones* y una ventana de transformación de tamaño *3x3*, donde puede observarse una gran diferencia entre ambos tiempos, llegando a ser hasta más de 7 veces menor (para otros tamaños de ventanas de transformación y coste y otras imágenes llega a ser hasta más de 10 veces menor) el tiempo invertido mediante la técnica de reutilización con respecto a la versión original del algoritmo. A la vista de estos datos, se podría concluir en que, en caso de utilizar el algoritmo *Census*, sería mucho más conveniente emplearlo con la aplicación de la técnica de reutilización de valores, con un tamaño de ventana de transformación de *3x3* (el más reducido) y un tamaño de ventana de coste más o menos elevado, ya que para una ventana de transformación reducida, el aumento en la ventana de coste no produce una diferencia tan pronunciada en el tiempo de ejecución requerido para el procesamiento, pudiendo así obtener la mayor tasa de acierto en el mejor tiempo posible.

No obstante, a pesar de todo el tiempo de ejecución sigue siendo ciertamente elevado, con lo cual será interesante estudiar los resultados obtenidos de las pruebas realizadas para el resto de algoritmos basados en *Hamming*.

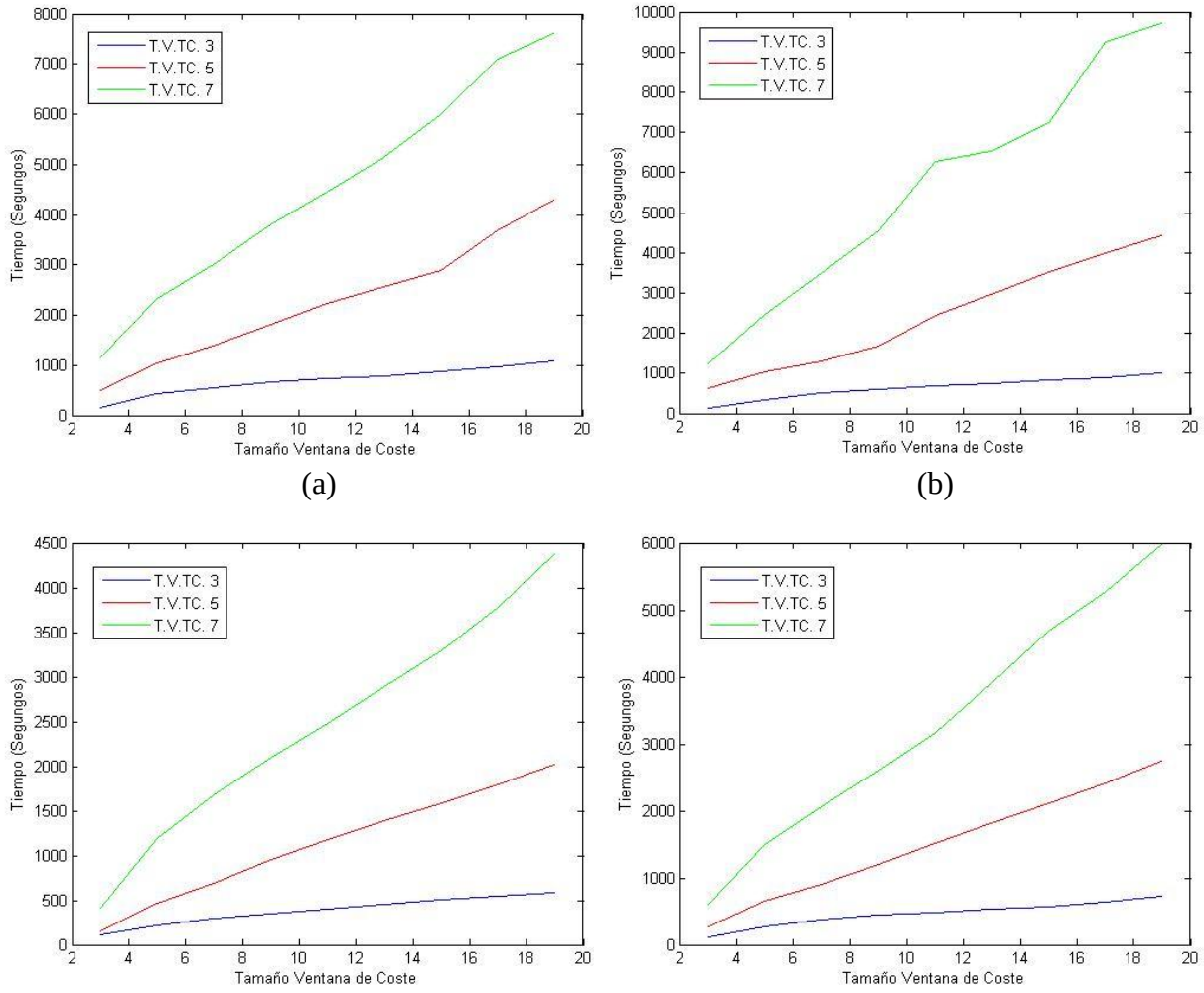


Figura 3.2.2. Gráficas comparativas del tiempo de ejecución para *Cones* (a), *Teddy* (b), *Tsukuba* (c) y *Venus* (d) frente al tamaño de la ventana de coste y ventana de transformación para el algoritmo *Census con reutilización parcial por columnas*.

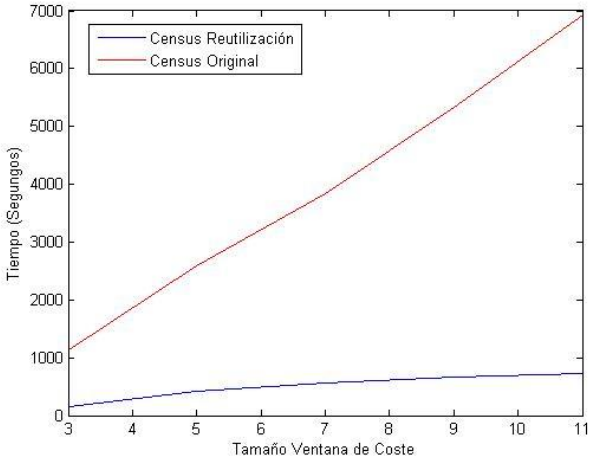
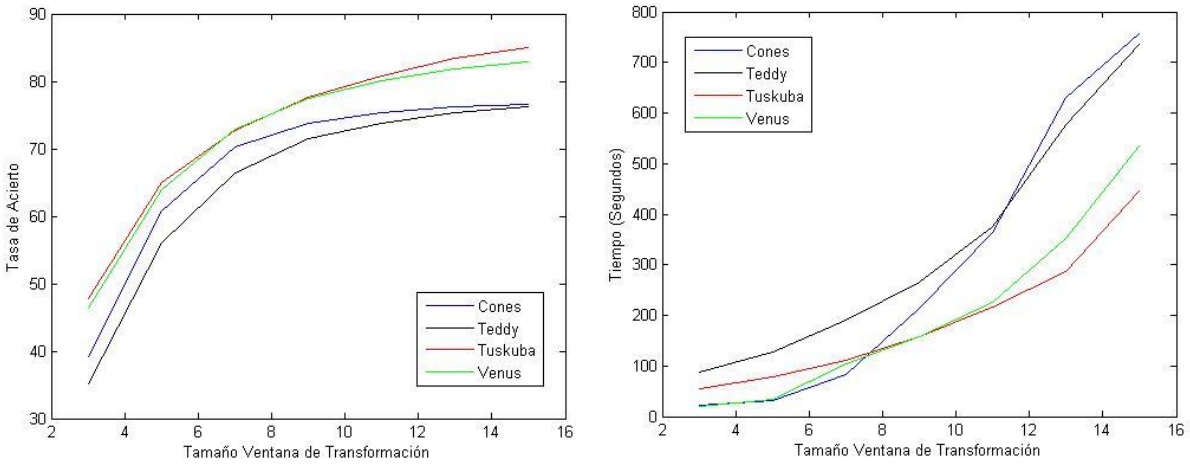


Figura 3.2.3. Gráfica comparativa del tiempo de ejecución para *Cones*, frente al tamaño de la ventana de coste para los algoritmos *Census con reutilización parcial por columnas* y *Census original* para una ventana de transformación de *3x3*.

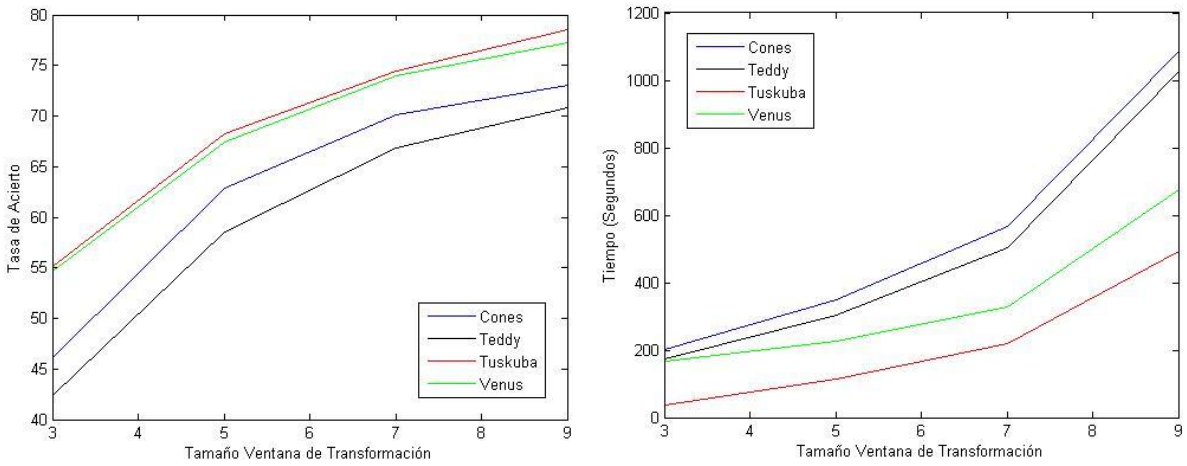
Para el algoritmo *mini-Census* únicamente entra en juego la ventana de transformación, ya que, tal y como se explicó en el capítulo anterior, en este algoritmo no se comparan pares de ventanas para el cálculo del coste, por lo cual aquí no ha lugar el tamaño de ventana de coste, sino solamente el de transformación. La *figura 3.2.4* muestra de forma conjunta la evolución tanto del porcentaje de acierto como del tiempo de ejecución de este algoritmo para cada una de las imágenes de *Middlebury* en función de la variación de la ventana de transformación:



(a) (b)

Figura 3.2.4. Gráficas comparativas de la tasa de acierto (a) y tiempo de ejecución (b) para cada una de las imágenes de *Middlebury* según el algoritmo *mini-Census*.

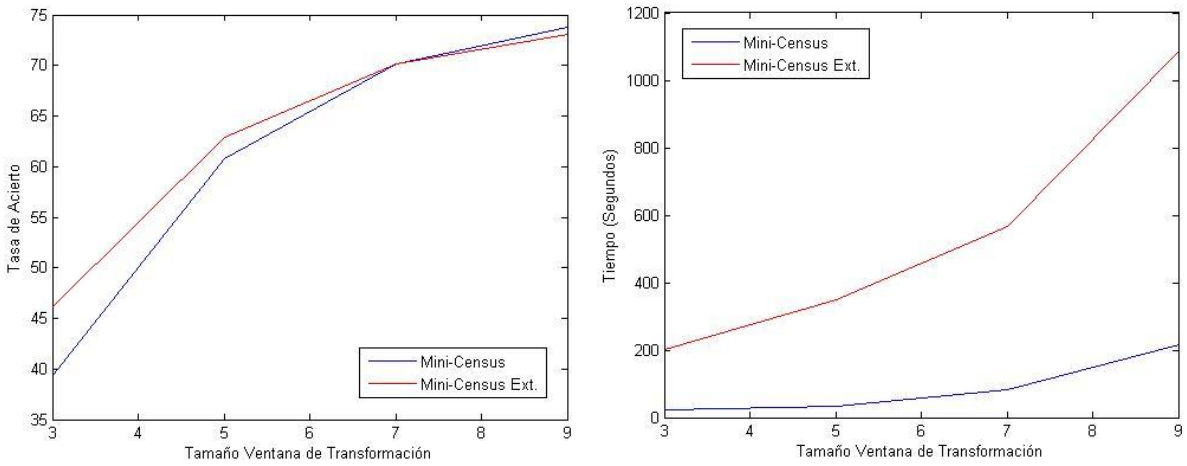
Y lo mismo sucede en el caso de *mini-Census Extendido*, cuyos resultados se muestran en la *figura 3.2.5*:



(a) (b)

Figura 3.2.5. Gráficas comparativas de la tasa de acierto (a) y tiempo de ejecución (b) para cada una de las imágenes de *Middlebury* según el algoritmo *mini-Census Extendido*.

Si se analizan detenidamente los resultados obtenidos para ambos algoritmos, comparándolos entre sí, es fácil observar (por ejemplo para el caso de la imagen *Cones*) lo que se muestra a continuación en la *figura 3.2.6*:



(a) (b)

Figura 3.2.6. Gráficas comparativas de la tasa de acierto (a) y tiempo de ejecución (b) para la imagen *Cones* según los algoritmos *mini-Census* y *mini-Census Extendido*.

Se puede apreciar que no existe una gran diferencia en la tasa de acierto conseguida por ambos algoritmos, todo lo contrario que ocurre con el tiempo de ejecución donde sí se refleja una diferencia bastante amplia, siendo mucho menor el tiempo consumido por el algoritmo *mini-Census* con respecto a *mini-Census Extendido*. Las *tablas 3.2.1 y 3.2.2* muestran los valores exactos de los datos representados en las gráficas de la *figura 3.2.6*.

	Tamaño Ventana de Coste			
	3x3	***5x5***	***7x7***	***9x9***
Mini-C	32.29%	60.82%	70.20%	73.81%
Mini-C Ext.	46.07%	62.92%	70.12%	73.07%

Tabla 3.2.1. Tasa de acierto respecto al tamaño de la ventana de transformación para los algoritmos *mini-Census* y *mini-Census Extendido* para la imagen *Cones*.

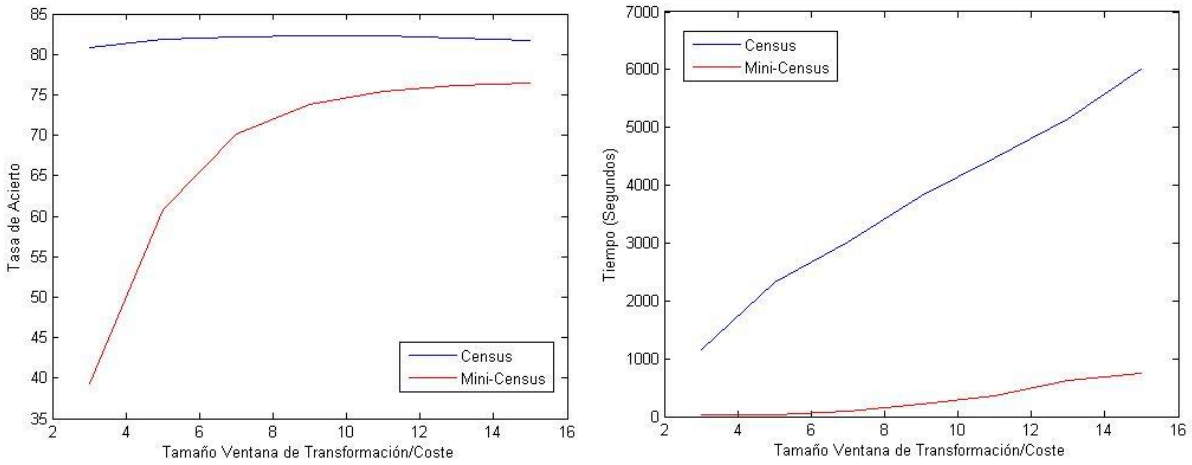
	Tamaño Ventana de Coste			
	3x3	***5x5***	***7x7***	***9x9***
Mini-C	21.66	32.46	82.72	214.54
Mini-C Ext.	202.1	350.4	565.8	1085.7

Tabla 3.2.2. Tiempo de ejecución (expresado en segundos) respecto al tamaño de la ventana de transformación para los algoritmos *mini-Census* y *mini-Census Extendido* para la imagen *Cones*.

Observando los valores exactos se comprueba que (salvo para el tamaño de ventana más pequeño donde la diferencia en la tasa de acierto es de algo más de unos 13 puntos porcentuales), la mayor diferencia en la tasa de acierto de ambos algoritmos apenas supera el 2% e incluso para el tamaño de ventana mayor el algoritmo *mini-Census* llega a superar a *mini-Census Extendido*. Sin embargo, el tiempo de ejecución para el algoritmo *mini-Census* es de entre 5 y 10 veces menor que para *mini-Census Extendido*. Este contraste entre los resultados de ambos algoritmos se reproduce igualmente para las otras imágenes.

A la vista de estos resultados, se podría concluir en que resulta más conveniente hacer uso del algoritmo *mini-Census* en lugar de *mini-Census Extendido* ya que ambos producen una tasa de acierto muy similar siendo el coste computacional del primero mucho menor que el del segundo.

Si se realiza una comparación entre el algoritmo *Census* en su versión original y el algoritmo *mini-Census* se puede apreciar lo mostrado a continuación en la *figura 3.2.7*:



(a) (b)

Figura 3.2.7. Gráficas comparativas de la tasa de acierto (a) y tiempo de ejecución (b) para la imagen *Cones* según los algoritmos *Census* y *mini-Census*.

En la gráfica de la izquierda (a), se puede observar el porcentaje de acierto conseguido por el algoritmo *Census* en función de la variación de la ventana de coste para una ventana de transformación de *7x7*, frente a la tasa de acierto obtenida por el algoritmo *mini-Census* en función de la variación de la ventana de transformación para el caso de la imagen *Cones*, mientras que en la gráfica de la derecha (b) se aprecia el tiempo de ejecución expresado en segundos para sendos algoritmos, siendo el del caso de *Census* el obtenido mediante la técnica de reutilización de valores.

Los resultados muestran que para los tamaños de ventana más pequeños, la diferencia en la tasa de acierto es bastante amplia, pero ésta se ve reducida conforme se aumenta la ventana, llegándose a reducir dicha diferencia hasta tan solo un 5% para el tamaño de ventana de *15x15*. Por otra parte, las diferencias en los tiempos de ejecución son mucho más marcadas, siendo muchísimo más reducido el tiempo de ejecución del algoritmo *mini-Census* con respecto al obtenido por *Census* en su versión original mediante la técnica de reutilización de valores. En el caso del tamaño de ventana de *15x15* el tiempo de ejecución obtenido por *mini-Census* es 8 veces menor con respecto al de *Census*.

Estos datos podrían ser indicativos de que quizás sea más conveniente emplear el algoritmo *mini-Census* con tamaños de ventana de transformación elevados en lugar del algoritmo *Census* original. Es interesante por tanto observar los resultados que se obtienen en la tasa de acierto del algoritmo *mini-Census* si se aplica el filtro de la mediana en la etapa de post-procesado, los cuales se muestran a continuación en la *figura 3.2.8*:

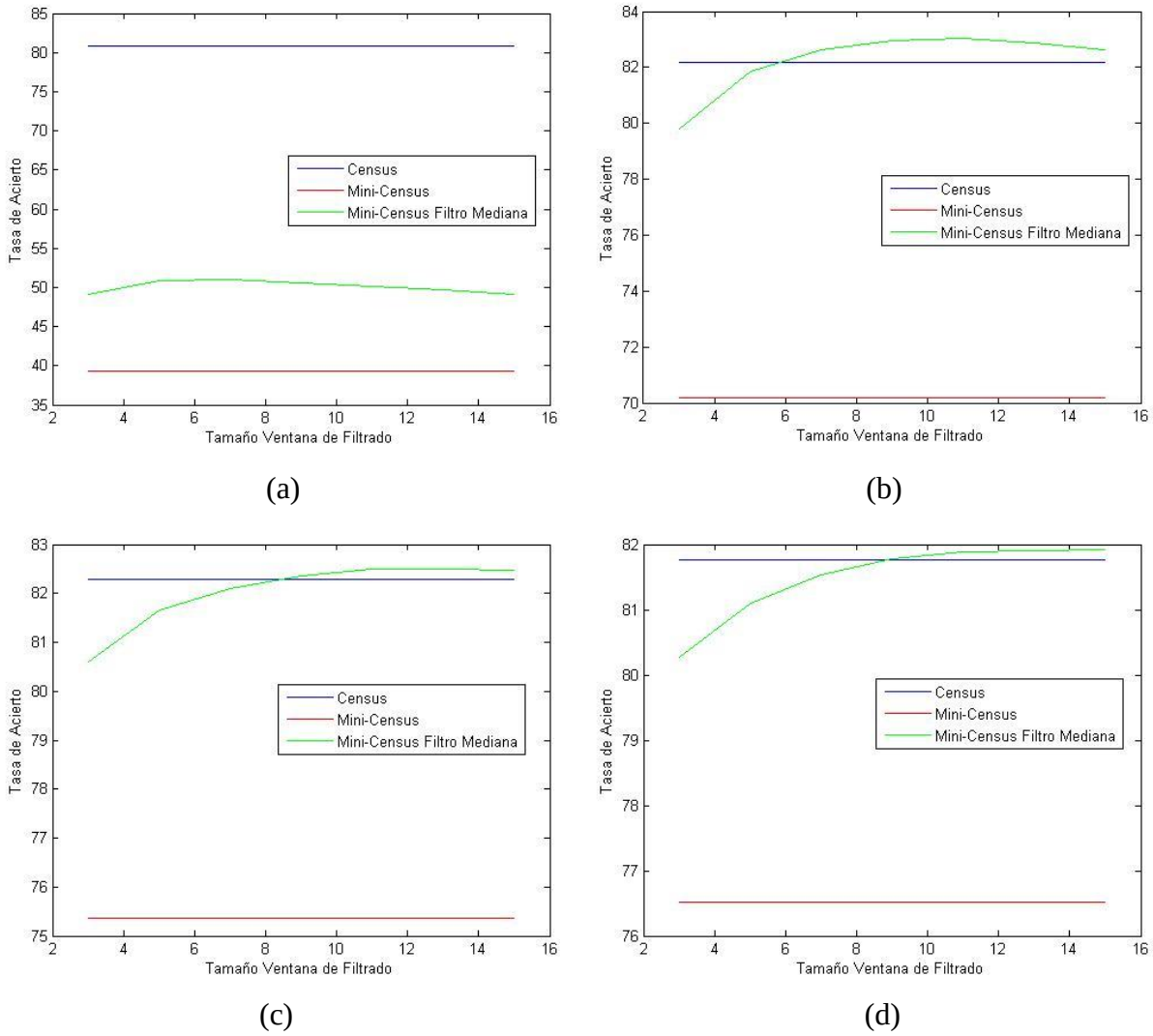


Figura 3.2.8. Gráficas comparativas de la tasa de acierto para la imagen *Cones* según los algoritmos *Census*, *mini-Census* y *mini-Census* con aplicación del filtro de la mediana en la etapa de post-procesado.

Las gráficas muestran el valor del porcentaje de acierto obtenido mediante los algoritmos *Census* y *mini-Census* con un tamaño de ventana de coste y transformación respectivamente de *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d), y se compara dicho valor con el obtenido al aplicar el filtro de mediana en la etapa de post-procesado al algoritmo *mini-Census*, realizando una variación del tamaño de ventana de filtrado desde *3x3* hasta *15x15*.

Se puede observar fácilmente que la tasa de acierto obtenida por *mini-Census* al añadirle el filtro de mediana en la etapa post-procesado se incrementa y mejora de tal forma que llega a superar la tasa de acierto obtenida por *Census* en su versión original a partir de un tamaño de ventana de transformación de *7x7* y un tamaño medio (entre *7x7* y *11x11*) de ventana de filtrado. La mejora conseguida llega en ocasiones a superar los

10 puntos porcentuales con respecto a la tasa obtenida por *mini-Census* sin la aplicación del filtro.

A la vista de estos resultados, se podría concluir en que sería preferible emplear la combinación del algoritmo *mini-Census* con la aplicación del filtro de mediana como etapa de post-procesado, ya que, aunque el algoritmo *mini-Census* consigue una tasa de acierto algo inferior a la obtenida mediante la versión original de *Census*, con la ayuda del post-filtrado se salva esta situación llegando incluso a superarla utilizando tamaños de ventanas de transformación y de filtrado relativamente pequeños, con lo cual se aprovecha la gran ventaja que ofrece *mini-Census*, la cual reside en un coste computacional sumamente inferior al de *Census* con lo que se obtendrían los mismos o incluso mejores porcentajes de acierto en un tiempo mucho menor.

A continuación se procede a analizar los resultados proporcionados por el algoritmo *Census Disperso*. En primer lugar se detallan los resultados obtenidos cuando el patrón de dispersión se aplica sobre la ventana de transformación y en segundo lugar se mostrarán los resultados cuando el patrón disperso se aplica sobre la ventana de coste. Cabe mencionar que los resultados mostrados son para el caso particular de la imagen *Cones*, si bien se ha decidido escoger esta imagen a modo de ejemplo para mostrar los datos ya que el comportamiento aquí mostrado se reproduce de igual forma para el resto de imágenes de *Middlebury*.

Las pruebas para estos algoritmos han sido realizadas utilizando cuatro modelos de patrón diferentes, empleando en primer lugar uno de 2 píxeles, en segundo lugar uno de 4 píxeles, en tercer lugar un patrón de 5 píxeles y por último un patrón que se ha denominado patrón *en cruz* ya que este patrón emplea los píxeles de la fila y la columna que pasan por el píxel central de la ventana empleada, formando así una cruz. El patrón de 2 píxeles utiliza un píxel situado a la izquierda del central y otro situado a la derecha del mismo, mientras que el patrón de 4 píxeles añade a esos dos otro píxel situado sobre el central y un cuarto situado bajo el central. El patrón de 5 píxeles emplea los cuatro píxeles situados en las esquinas de la ventana y el píxel central. Los tamaños de ventana de trabajo empleados para las pruebas son ventanas de transformación desde *3x3* hasta *9x9* y ventanas de coste de *3x3*, *5x5* y *7x7*.

A continuación la *figura 3.2.9* muestra visualmente la configuración de estos cuatro patrones para un tamaño de ventana de trabajo de *5x5*:

0	0	0	0	0
0	0	0	0	0
1	0	0	0	1
0	0	0	0	0
0	0	0	0	0

(a)

0	0	1	0	0
0	0	0	0	0
1	0	0	0	1
0	0	0	0	0
0	0	1	0	0

(b)

1	0	0	0	1
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
1	0	0	0	1

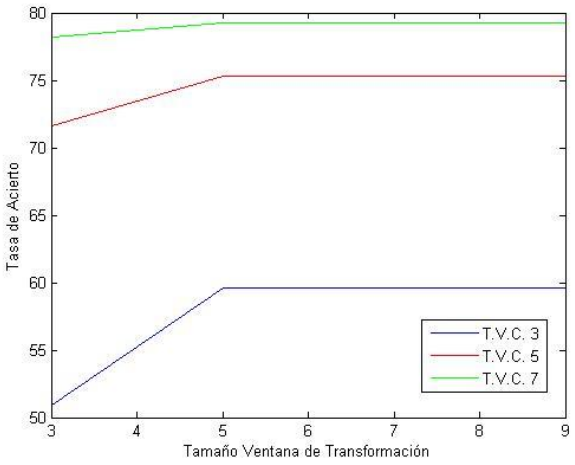
(c)

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

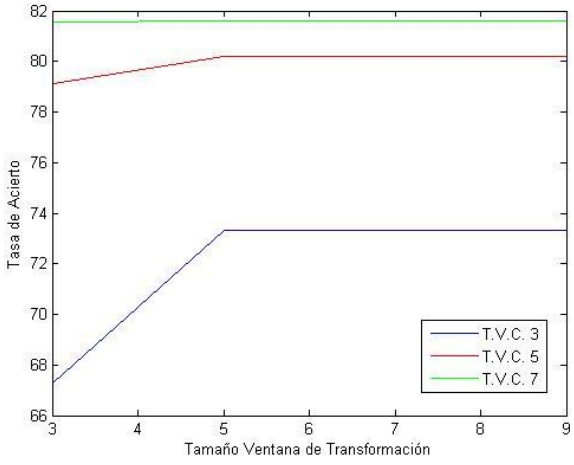
(d)

Figura 3.2.9. Patrones de dispersión de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y patrón en cruz (d) aplicados en las pruebas realizadas mediante el algoritmo *Census Disperso* para un tamaño de ventana de *5x5*.

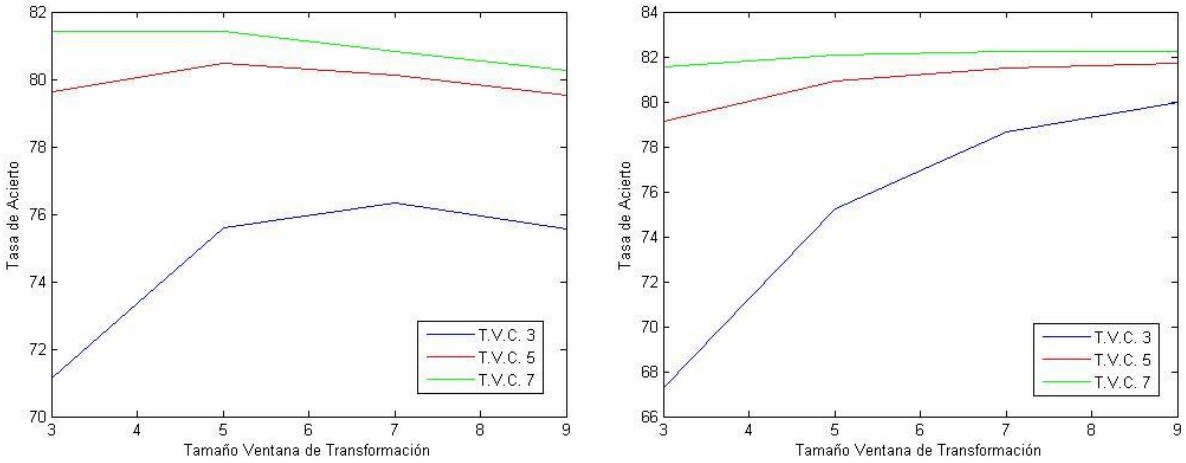
La *figura 3.2.10* muestra los resultados obtenidos referentes a la tasa de acierto cuando el patrón disperso se aplica sobre la ventana de transformación, para una variación de la ventana de transformación desde *3x3* hasta *9x9* y tamaños de ventana de coste de *3x3*, *5x5* y *7x7* para cada uno de los cuatro patrones que se han empleado.



(a)



(b)



(c) (d)

Figura 3.2.10. Gráficas comparativas de la tasa de acierto para la imagen *Cones* según el algoritmo *Census Disperso*, con aplicación del patrón de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y en cruz (d) sobre la ventana de transformación para tamaños de ventana de coste de *3x3*, *5x5* y *7x7*.

La evolución de la tasa de acierto es diferente según el patrón empleado. En el caso de los patrones de 2 y 4 píxeles se puede apreciar que, salvo para el primer tamaño de ventana de transformación (*3x3*), la tasa de acierto es lineal, no varía a pesar del aumento del tamaño de ventana de transformación. Esto sucede porque, a partir del tamaño de ventana de transformación de *5x5* los píxeles que se van a contemplar para el procesado no cambian. Aunque se aumente el tamaño de ventana de transformación, los píxeles que se emplearán para el procesado son siempre los mismos, los que están situados justo antes de los píxeles contiguos al central, por lo que siempre aportan la misma información. Sin embargo, para el tamaño de *3x3* esto no es posible ya que solamente se pueden utilizar los que están contiguos al central porque no hay más píxeles disponibles.

Para el caso del patrón de 5 píxeles la probabilidad de acierto obtenida llega incluso a disminuir con el aumento de la ventana de transformación. Esto podría ser debido a que, según la configuración de dicho patrón, al aumentar el tamaño de la ventana, los 4 píxeles que se sitúan siempre en las esquinas de la misma están cada vez más alejados del píxel central y por lo tanto el ratio de información que aportan con respecto al total que alberga la ventana es cada vez menor. Sería necesario ir aumentando también los píxeles empleados para el procesado en posiciones medias entre los extremos y el píxel central a la par que se aumenta el tamaño de la ventana de transformación para que la evolución en la tasa de acierto fuera siempre en aumento.

La evolución mostrada por el patrón en cruz es siempre creciente, lo cual era de esperar ya que, por su configuración, aunque aumenta el tamaño de la ventana de

transformación también aumenta a su vez el número de píxeles empleados para el procesado.

En cuanto a la influencia aportada por la variación de la ventana de coste, es fácil observar que, independientemente del patrón utilizado, la tasa de acierto crece conforme al empleo de tamaños de ventana de coste mayores, siendo dicho crecimiento más pronunciado entre los tamaños menores (*3x3* y *5x5*), viéndose reducida esta diferencia para tamaños mayores. Particularmente, en los resultados mostrados la mayor tasa de acierto se logra con la ventana de coste de *7x7*, sin embargo es interesante observar los datos más detalladamente, los cuales se muestran a continuación en la *tabla 3.2.3*:

	Tamaño Ventana de Transformación			
	3x3	***5x5***	***7x7***	***9x9***
2 píxeles	71.16%	**79.25%**	79.25%	79.25%
4 píxeles	81.55%	**81.62%**	81.62%	81.62%
5 píxeles	**81.43%**	81.41%	80.83%	80.27%
Cruz	81.55%	82.08%	82.23%	**82.26%**

Tabla 3.2.3. Tasa de acierto para la imagen *Cones*, obtenida por el algoritmo *Census Disperso* cuando los patrones de 2 píxeles, 4 píxeles, 5 píxeles y en cruz se aplican sobre la ventana de transformación para un tamaño de ventana de coste de *7x7*.

Se ha resaltado la mayor tasa de acierto conseguida para cada uno de los patrones empleados. La mejor tasa es del 82.26%, conseguida por el patrón en cruz empleando la mayor ventana de transformación (*9x9*). Sin embargo, cabe mencionar que las mejores tasas de acierto para los patrones de 2 y 4 píxeles son, respectivamente, del 79.25% y del 81.62%, con lo cual la diferencia con respecto a la tasa obtenida por el patrón en cruz es, respectivamente, del 3.01% y del 0.64% y además han sido obtenidas empleando un tamaño de ventana de transformación de *5x5*. Este dato es muy importante ya que indica que para una ventana de coste relativamente amplia (como es en este caso la de *7x7*), las tasas de acierto conseguidas por los distintos patrones no distan mucho entre sí, y es perfectamente plausible utilizar un patrón de tan solo 2 o 4 píxeles sacrificando un porcentaje mínimo de probabilidad de acierto y ganando así la ventaja en la eficiencia ya que cuantos menos píxeles se empleen en el procesado menor será el tiempo de ejecución requerido por el algoritmo.

Los resultados en lo que a la eficiencia del algoritmo se refiere se muestran a continuación en la *figura 3.2.11*:

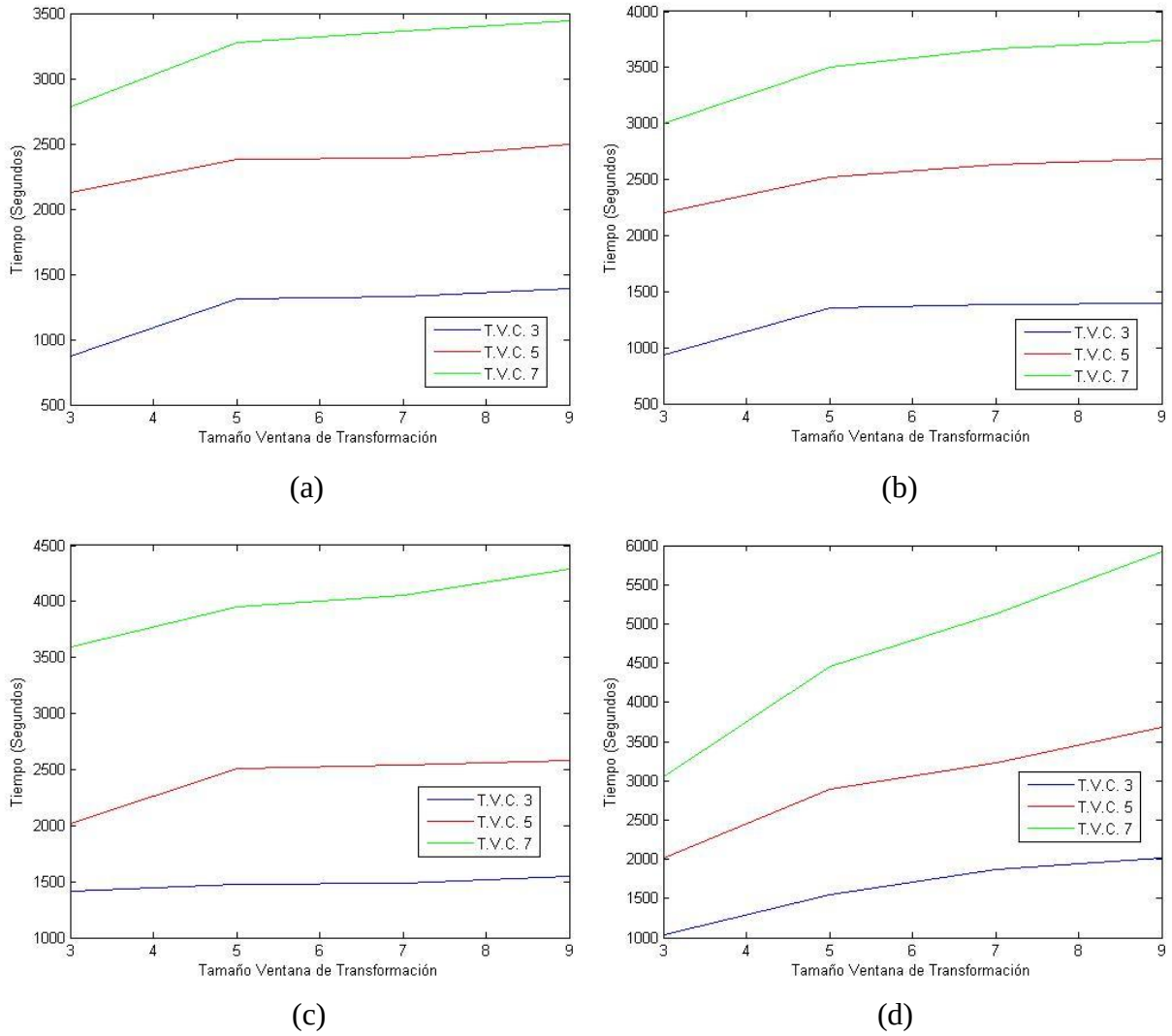


Figura 3.2.11. Gráficas comparativas del tiempo de ejecución para la imagen *Cones* según el algoritmo *Census Disperso*, con aplicación del patrón de 2 píxeles (a),4 píxeles (b), 5 píxeles (c) y en cruz (d) sobre la ventana de transformación para tamaños de ventana de coste de *3x3*, *5x5* y *7x7*.

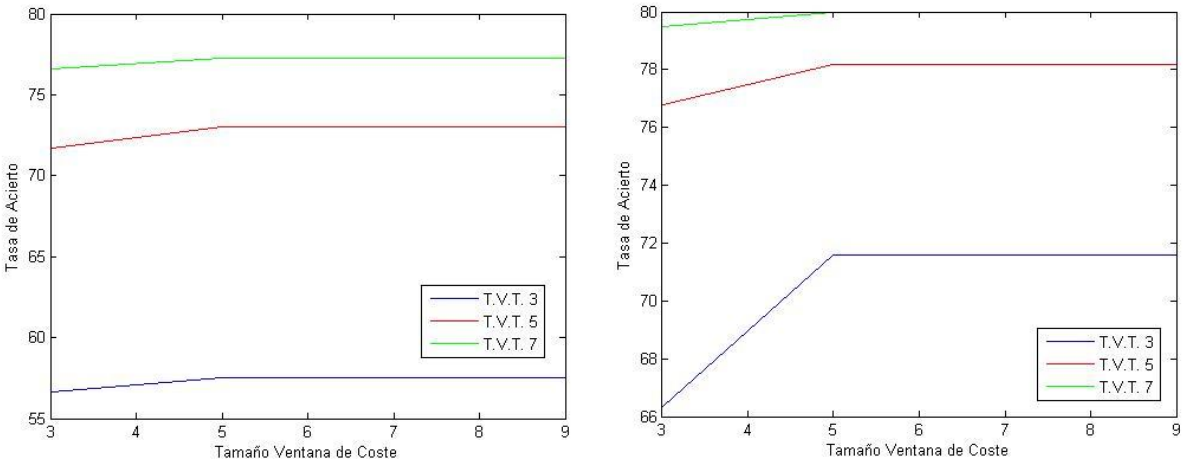
Como cabía esperar, el tiempo de ejecución aumenta tanto al aumentar la ventana de transformación como la ventana de coste. A continuación, la *tabla 3.2.4* muestra los valores exactos para la ventana de coste de *7x7*:

	Tamaño Ventana de Transformación			
	3x3	***5x5***	***7x7***	***9x9***
2 píxeles	2784.8	**3279.5**	3359.7	3439.4
4 píxeles	3002	**3505.3**	3666.5	3737.3
5 píxeles	**3597.6**	3948.9	4058.2	4288.4
Cruz	3055.5	4464.2	5135.7	**5915.6**

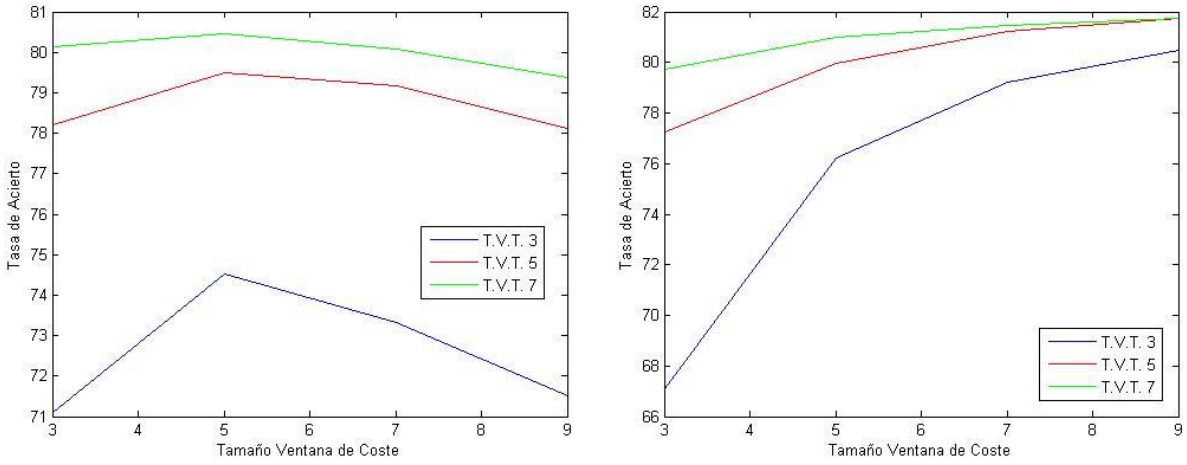
Tabla 3.2.4. Tiempo de ejecución (en segundos) para *Cones* mediante *Census Disperso*.

Se han resaltado los valores para los cuales se obtiene la mayor tasa de acierto con cada patrón. A la vista de los datos, se puede confirmar que, como ya se mencionó al observar los resultados en las tasas de acierto del algoritmo, es preferible utilizar un patrón de 2 píxeles con una ventana de transformación reducida para ejecutar el algoritmo, ya que se obtiene así un porcentaje de acierto prácticamente idéntico al que se obtendría empleando el patrón en cruz con la mayor ventana de transformación en casi la mitad de tiempo. No obstante se trata de unos tiempos de ejecución ciertamente elevados, con lo cual será interesante analizar los resultados proporcionados por el algoritmo cuando el patrón de dispersión se aplica sobre la ventana de coste en lugar de sobre la ventana de transformación.

A continuación la *figura 3.2.12* muestra la tasa de acierto obtenida al aplicar cada uno de los cuatro patrones sobre la ventana de coste, para una variación en el tamaño de la ventana de coste desde *3x3* hasta *9x9* y ventanas de transformación de *3x3*, *5x5* y *7x7*:



(a) (b)



(c) (d)

Figura 3.2.12. Gráficas comparativas de la tasa de acierto para la imagen *Cones* según el algoritmo *Census Disperso*, con aplicación del patrón de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y en cruz (d) sobre la ventana de coste para tamaños de ventana de transformación de *3x3, 5x5* y *7x7*.

Se puede observar que el comportamiento en lo que a la tasa de acierto se refiere es prácticamente idéntico al estudiado anteriormente cuando se aplican los patrones sobre la ventana de transformación.

En el caso de los patrones de 2 y 4 píxeles la tasa de acierto obtenida no varía a partir del tamaño de *5x5* debido a la configuración de los mismos. De igual forma, en el patrón de 5 píxeles la tasa de acierto disminuye al aumentar la ventana de coste mientras que para el patrón en cruz aumenta, debido también a las configuraciones de dichos patrones, ya comentadas anteriormente.

De nuevo, se aprecia claramente que al aumentar el tamaño de la ventana de transformación la tasa de acierto se ve también aumentada siendo menos pronunciada la diferencia para valores altos (de *5x5* a *7x7*) que para los valores más pequeños (de *3x3* a *5x5*). La *tabla 3.2.5* muestra los valores exactos mostrados en las gráficas para la ventana de transformación de *7x7* obtenidos con cada uno de los patrones empleados.

	Tamaño Ventana de Coste			
	3x3	***5x5***	***7x7***	***9x9***
2 píxeles	76.58%	**77.28%**	77.28%	77.28%
4 píxeles	79.47%	**79.96%**	79.96%	79.96%
5 píxeles	80.14%	**80.46%**	80.08%	79.39%
Cruz	79.73%	80.99%	81.44%	**81.75%**

Tabla 3.2.5. Tasa de acierto para *Cones* mediante *Census Disperso* para un tamaño de ventana de transformación de *7x7*.

Se han resaltado los valores más altos que se han obtenido con cada uno de los distintos patrones, resultando nuevamente el mejor el conseguido por el patrón en cruz con un valor del 81.75%. Sin embargo, una vez más se aprecia que la diferencia con respecto a la tasa más alta obtenida por los otros patrones que utilizan un menor número de píxeles no es demasiado marcada, de hecho no llega a superar el 4.5% respecto a la lograda por el patrón de 2 píxeles y el 1.3% respecto al patrón de 5 píxeles, lo cual vuelve de nuevo a ser indicativo de que podría ser preferible usar un patrón con el menor número de píxeles posible sacrificando un mínimo porcentaje de la tasa de acierto para ganar en eficiencia.

Los tiempos de ejecución asociados a estos resultados se muestran gráficamente a continuación en la *figura 3.2.13*:

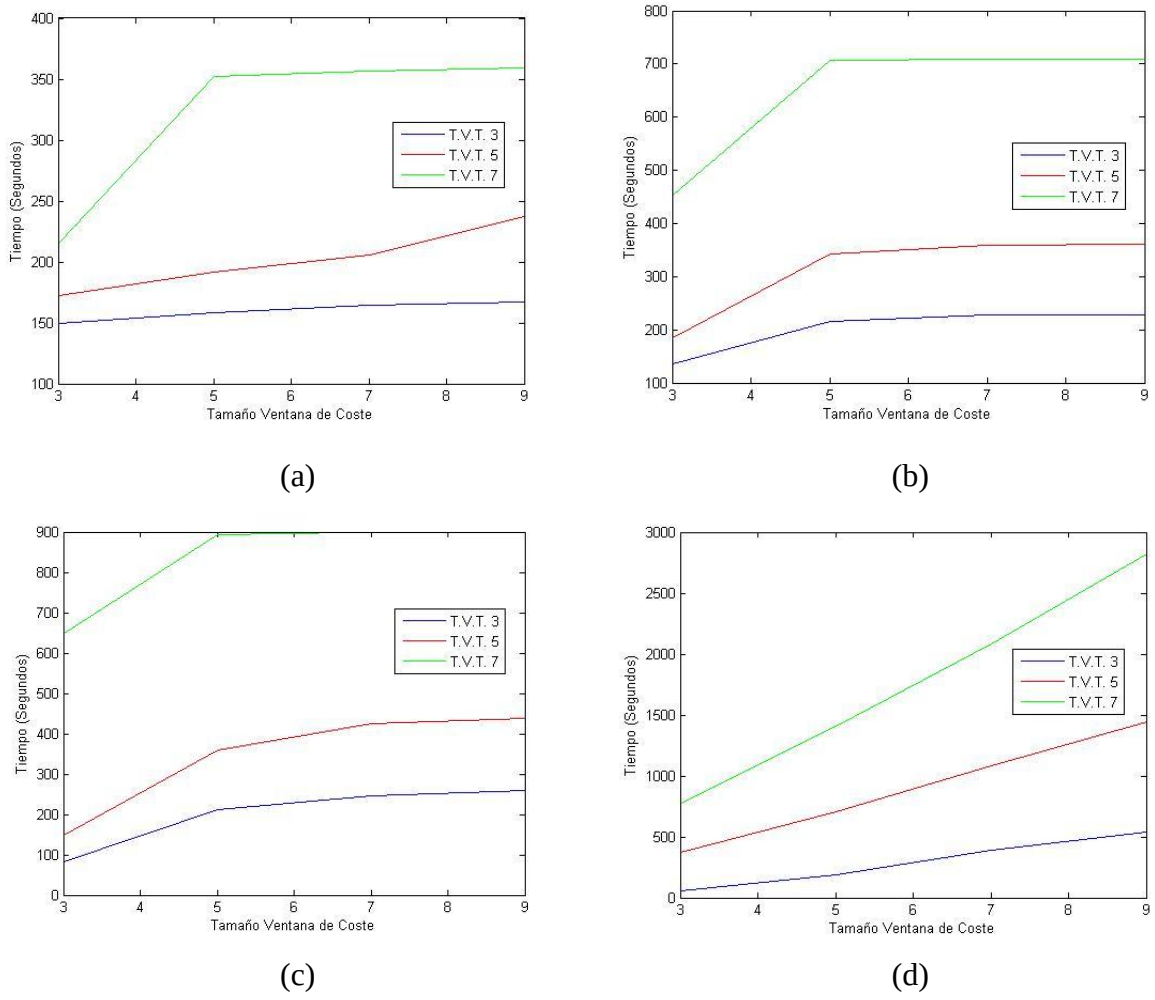


Figura 3.2.13. Gráficas comparativas del tiempo de ejecución para la imagen *Cones* según el algoritmo *Census Disperso*, con aplicación del patrón de 2 píxeles (a),4 píxeles (b), 5 píxeles (c) y en cruz (d) sobre la ventana de coste para tamaños de ventana de transformación de *3x3*, *5x5* y *7x7*.

De nuevo como cabía esperar el tiempo de ejecución crece conforme se aumenta el tamaño tanto de la ventana de transformación como de la ventana de coste. Los valores exactos para el tamaño de ventana de transformación de *7x7* son los que se muestran en la *tabla 3.2.6*.

	Tamaño Ventana de Coste			
	3x3	***5x5***	***7x7***	***9x9***
2 píxeles	214.9	**351.9**	356.3	359.4
4 píxeles	453.1	**705.8**	708.9	709.2
5 píxeles	647	**893.2**	899.3	899.8
Cruz	773.1	1411.7	2086.4	**2822.6**

Tabla 3.2.6. Tiempo de ejecución (expresado en segundos) para la imagen *Cones*, obtenido por el algoritmo *Census Disperso* cuando los patrones de 2 píxeles, 4 píxeles, 5 píxeles y en cruz se aplican sobre la ventana de coste para un tamaño de ventana de transformación de *7x7*.

A la vista de estos resultados, una vez más, tal como ocurría en el caso en el que los patrones se aplicaban sobre la ventana de transformación, se confirma que es más conveniente usar un patrón con el menor número de píxeles posible ya que se obtiene una tasa de acierto muy similar a la obtenida con el patrón en cruz en un tiempo hasta 8 veces menor (como es el caso del patrón en cruz con respecto al patrón de 2 píxeles), empleando una ventana de transformación de *7x7* y una ventana de coste pequeña (*5x5*).

Es importante notar que el tiempo de ejecución obtenido en este caso es bastante inferior al requerido por el algoritmo cuando los patrones se aplican sobre la ventana de transformación. Este hecho se puede observar gráficamente a continuación en la *figura 3.2.14*:

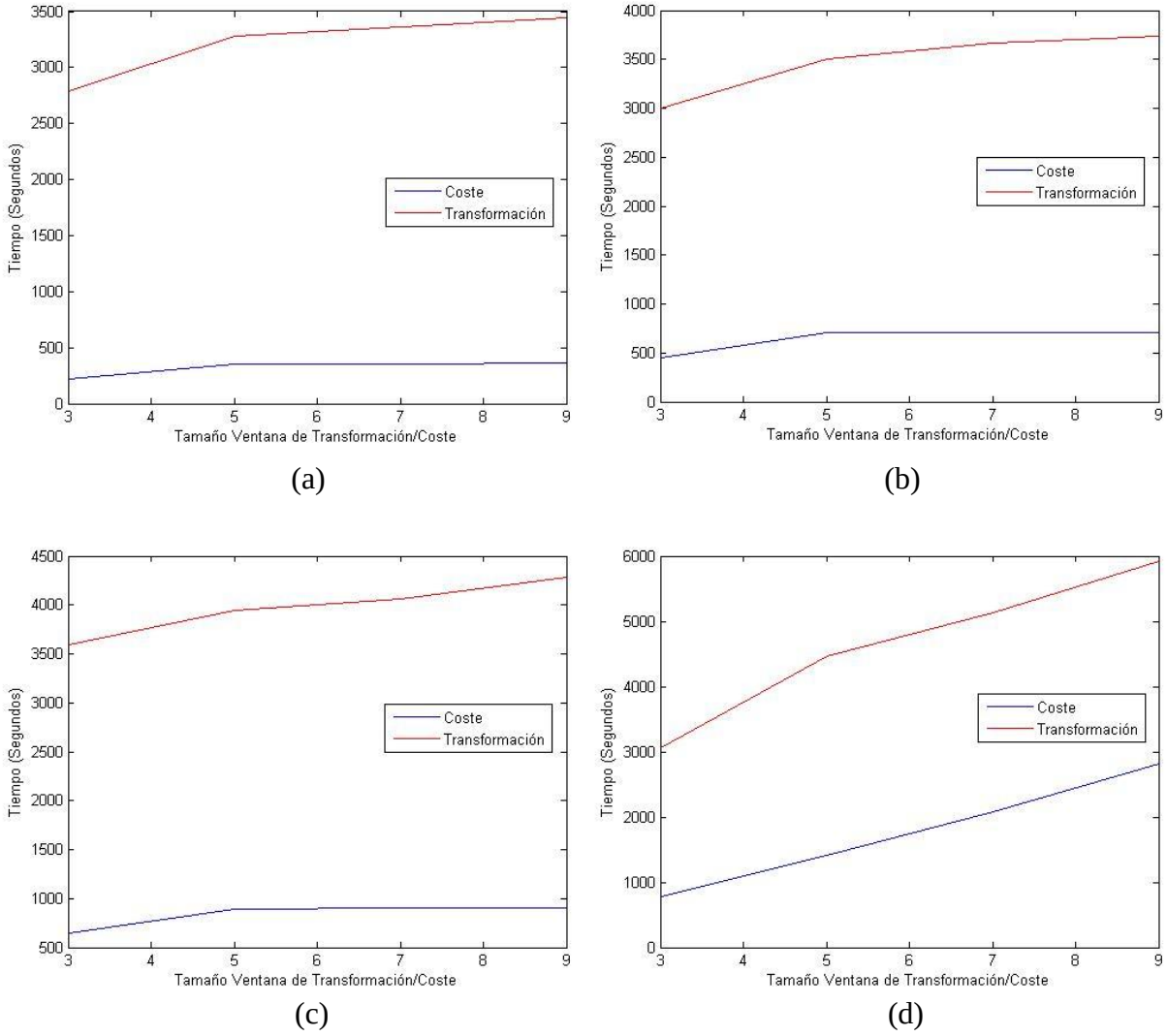


Figura 3.2.14. Gráficas comparativas del tiempo de ejecución para la imagen *Cones* según el algoritmo *Census Disperso*, con aplicación del patrón de 2 píxeles (a),4 píxeles (b), 5 píxeles (c) y en cruz (d) sobre la ventana de coste y sobre la ventana de transformación para una tamaño de *7x7*.

La *tabla 3.2.7* muestra los valores exactos para el caso del patrón de 2 píxeles:

	Tamaño Ventana de Coste/Transformación			
	3x3	***5x5***	***7x7***	***9x9***
Transformación	2784.8	**3279.5**	3359.7	3439.4
Coste	214.9	**351.9**	356.3	359.4

Tabla 3.2.7. Tiempo de ejecución (expresado en segundos) para la imagen *Cones*, obtenido por el algoritmo *Census Disperso* cuando el patrón de 2 píxeles se aplica sobre la ventana de transformación y sobre la ventana de coste para un tamaño de *7x7*.

Se puede apreciar que hay una diferencia abismal en el tiempo de ejecución requerido según se aplique el patrón sobre la ventana de transformación o sobre la ventana de coste, siendo de hasta casi 10 veces menor cuando se emplea sobre esta última. Esta diferencia se debe a que el proceso que realmente consume la mayor parte del tiempo de ejecución es el proceso del cálculo del coste, no el de la realización de la *transformada Census* el cual es un proceso relativamente rápido. Por lo tanto, al aplicar el patrón sobre la ventana de transformación, el proceso del cálculo del coste se lleva a cabo teniendo en cuenta todos los píxeles de la ventana de coste con lo cual el impacto en el tiempo de ejecución es mucho menor que si se aplica el patrón sobre la ventana de coste, ya que en este último caso sí que se está reduciendo en gran medida el coste computacional puesto que es el proceso que consume prácticamente la totalidad del tiempo de ejecución.

Es por tanto indispensable analizar así mismo la diferencia producida en la tasa de acierto, la cual se muestra a continuación en la *figura 3.2.15*:

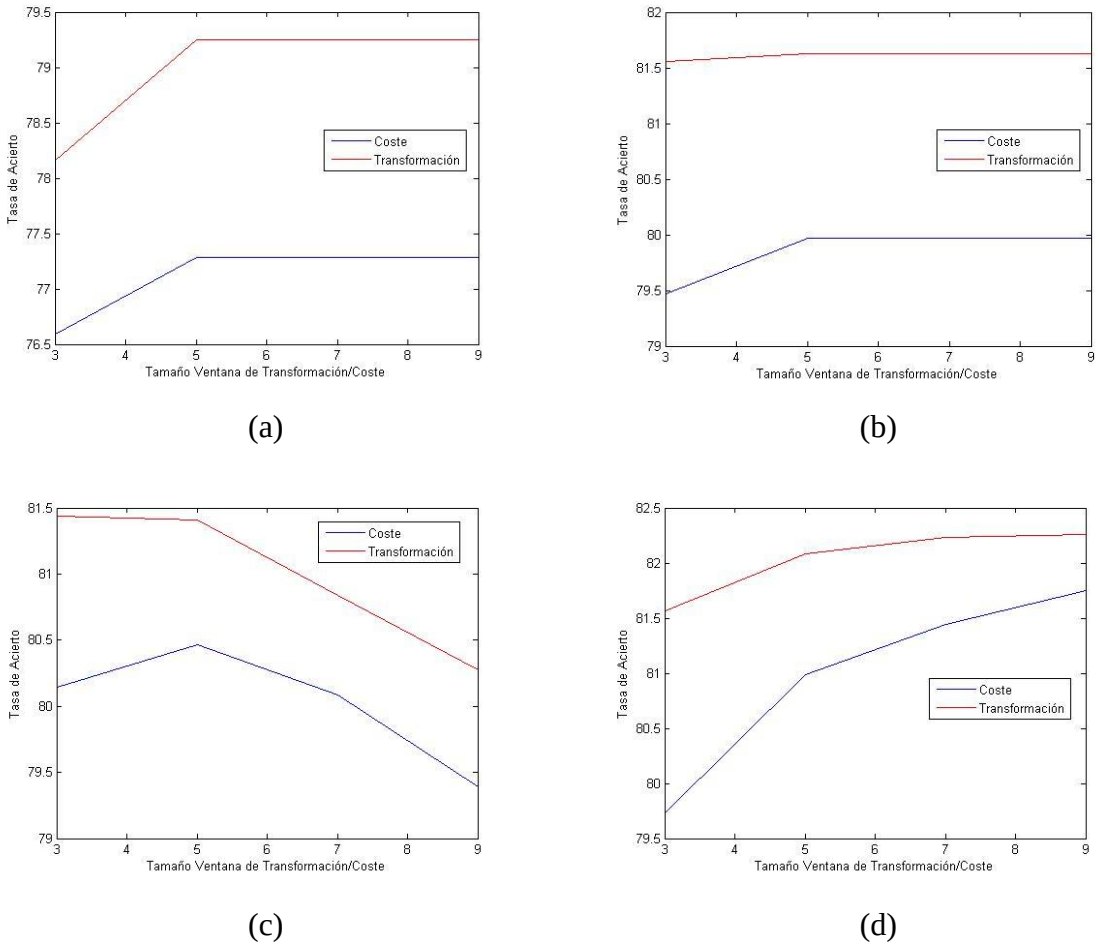


Figura 3.2.15. Gráficas comparativas de la tasa de acierto para la imagen *Cones* según el algoritmo *Census Disperso*, con aplicación del patrón de 2 píxeles (a),4 píxeles (b), 5 píxeles (c) y en cruz (d) sobre la ventana de coste y sobre la ventana de transformación para una tamaño de *7x7*.

La *tabla 3.2.8* muestra los valores exactos para el patrón de 2 píxeles:

	Tamaño Ventana de Coste/Transformación			
	3x3	***5x5***	***7x7***	***9x9***
Transformación	78.16%	**79.25%**	79.25%	79.25%
Coste	76.58%	**77.28%**	77.28%	77.28%

Tabla 3.2.8. Tasa de acierto para la imagen *Cones*, obtenida por el algoritmo *Census Disperso* cuando el patrón de 2 píxeles se aplica sobre la ventana de transformación y sobre la ventana de coste para un tamaño de *7x7*.

Se puede apreciar claramente que la diferencia en el porcentaje de acierto obtenido es mínima (no llega al 2% en el caso del patrón de 2 píxeles), con lo cual se puede concluir con rotundidad en que es más conveniente emplear el algoritmo *Census Disperso* aplicando el patrón de dispersión sobre la ventana de coste en lugar de sobre la ventana de transformación puesto que se obtiene una tasa de acierto prácticamente idéntica en un tiempo hasta casi 10 veces menor, y además es preferible utilizar un patrón con el menor número de píxeles activos posible (2 o 4 píxeles) pudiendo obtener así la mejor tasa de acierto con un gran impacto en la mejora del tiempo de ejecución requerido por el algoritmo para alcanzar dicha tasa. Las *tablas 3.2.9* y *3.2.10* muestran la prueba fehaciente de esta afirmación, en las que se muestran, respectivamente, las comparativas entre la máxima tasa de acierto conseguida por el algoritmo cuando se aplica el patrón sobre la ventana de coste y sobre la ventana de transformación para cada uno de los cuatro patrones propuestos y entre el tiempo de ejecución requerido en cada caso para alcanzar dicha tasa, pudiéndose apreciar una mínima diferencia en el porcentaje de acierto y una gran diferencia en el tiempo de ejecución.

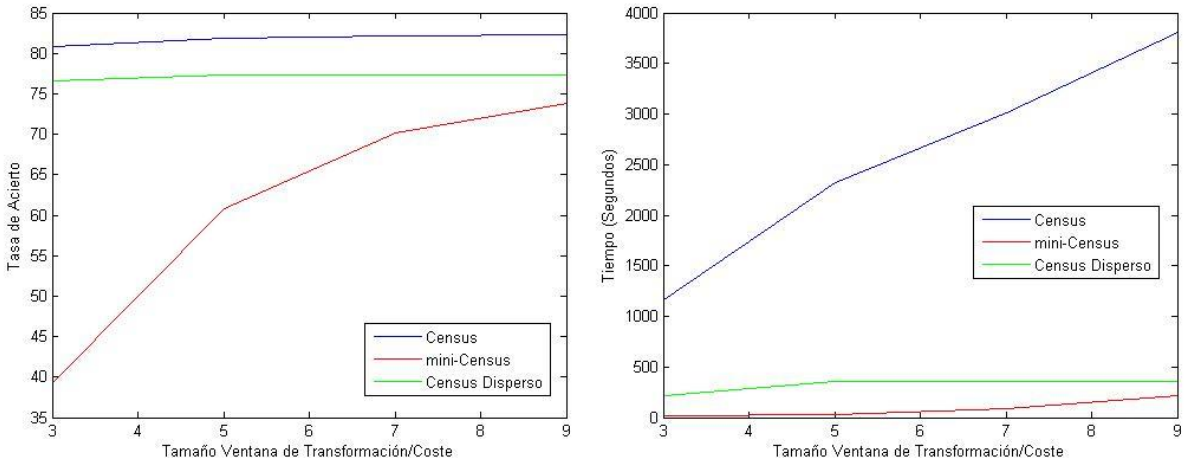
	Patrón Aplicado			
	2 Píxeles	***4 Píxeles***	***5 Píxeles***	***Cruz***
Transformación	79.25%	81.62%	81.43%	82.26%
Coste	77.28%	79.96%	80.46%	81.75%

Tabla 3.2.9. Comparativa de la máxima tasa de acierto para la imagen *Cones*, obtenida por el algoritmo *Census Disperso* para cada patrón cuando se aplica sobre la ventana de transformación y sobre la ventana de coste.

	Patrón Aplicado			
	2 Píxeles	***4 Píxeles***	***5 Píxeles***	***Cruz***
Transformación	3279.5	3505.3	3597.6	5915.6
Coste	351.9	705.8	893.2	2822.6

Tabla 3.2.10. Comparativa del tiempo de ejecución (expresado en segundos) para la imagen *Cones*, obtenida por el algoritmo *Census Disperso* para cada patrón cuando se aplica sobre la ventana de transformación y sobre la ventana de coste, empleado para lograr la máxima tasa de acierto posible.

Si se realiza la comparación entre los algoritmos *Census*, *mini-Census* y *Census Disperso* se pueden observar los resultados mostrados en la *figura 3.2.15* en lo referente a la tasa de acierto y tiempo de ejecución conseguidos por cada uno de ellos para la imagen *Cones*. En el caso del algoritmo *Census Disperso* los resultados mostrados se corresponden al patrón de 2 píxeles cuando se aplica sobre la ventana de coste para una ventana de transformación de *7x7*. En el caso del algoritmo *Census* los resultados mostrados se corresponden también con un tamaño de ventana de transformación de *7x7* y el tiempo de ejecución es el correspondiente al empleo de la técnica de reutilización de valores.



(a) (b)

Figura 3.2.16. Gráficas comparativas de la tasa de acierto (a) y tiempo de ejecución (b) para la imagen *Cones*, según los algoritmos *Census*, *mini-Census* y *Census Disperso*.

Se puede apreciar que la tasa de acierto conseguida por el algoritmo de *Census Disperso* aplicando un patrón de tan solo 2 píxeles sobre la ventana de coste no dista apenas de la conseguida por el algoritmo *Census* original y supera ampliamente (sobre todo para tamaños pequeños de ventana) a la obtenida mediante *mini-Census* y, observando los tiempos de ejecución, se puede ver que el tiempo consumido por *Census*

Disperso no dista demasiado del requerido por *mini-Census* y es mucho menor que el del algoritmo *Census* en su versión original incluso con del empleo de la técnica de reutilización de valores. Sería por tanto interesante observar los resultados obtenidos en la tasa de acierto al aplicar el filtro de mediana como etapa de post-filtrado sobre el algoritmo *Census Disperso*, los cuales se muestran en la *figura 3.2.17*.

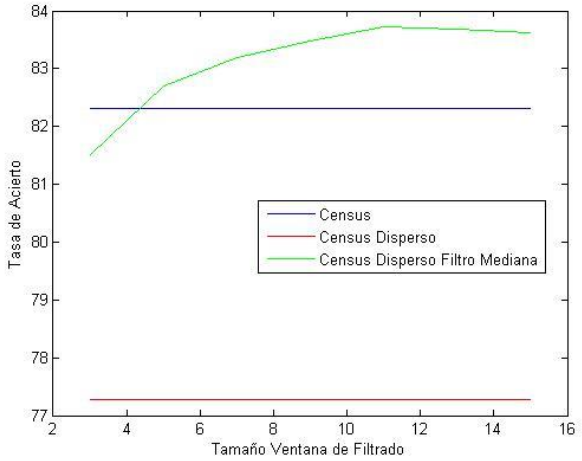


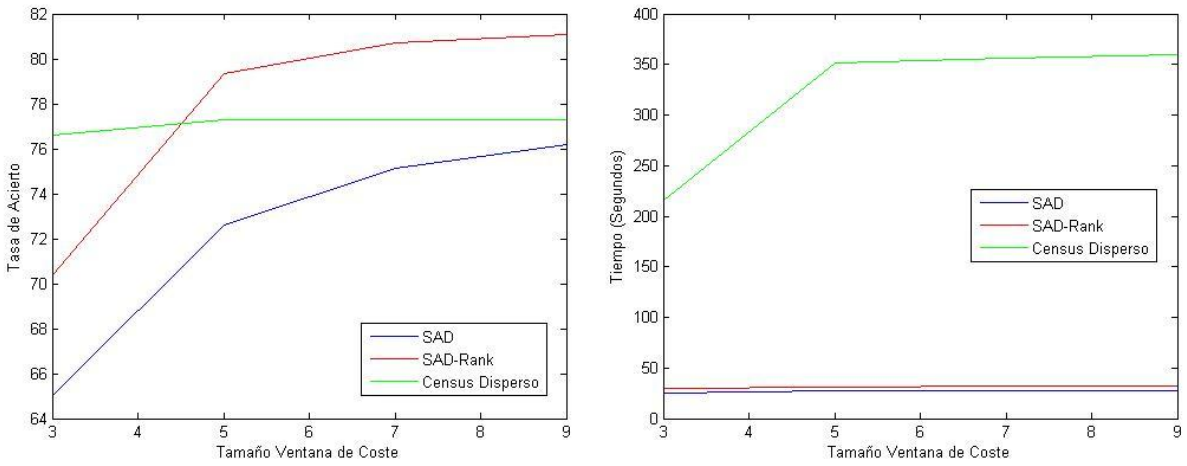
Figura 3.2.17. Gráfica comparativa de la tasa de acierto para la imagen *Cones* según los algoritmos *Census*, *Census Disperso* y *Census Disperso* con aplicación del filtro de la mediana en la etapa de post-procesado.

La figura muestra el valor de la máxima tasa de acierto del algoritmo *Census* original (azul), frente al valor de la máxima tasa de acierto conseguida por el algoritmo *Census Disperso* (rojo) y cómo esta última evoluciona al aplicar el filtro de la mediana con diferentes tamaños de ventana de filtrado (verde). Es importante notar cómo al aplicar el filtro de la mediana sobre el algoritmo *Census Disperso* la tasa de acierto se ve mejorada de tal forma que incluso se logra superar la obtenida mediante el algoritmo *Census* en su versión original, incluso para un tamaño de ventana de filtrado pequeño (como puede ser el de *5x5*).

Por lo tanto, a la vista de estos resultados, se podría afirmar que el algoritmo más conveniente a utilizar de todos los analizados que hacen uso del criterio de *Hamming* sería el algoritmo *Census Disperso* aplicando el patrón de dispersión sobre la ventana de coste y combinándolo con la aplicación de un filtro de mediana en la etapa de post-procesado, de tal forma que se puede obtener la mejor tasa de acierto posible aplicando un patrón de tan sólo 2 píxeles reduciendo así en gran medida el coste computacional generado por el algoritmo *Census*, con lo que se obtiene un tasa de acierto óptima en el menor tiempo de ejecución posible.

Analizados todos los algoritmos implementados durante la realización del proyecto, sería procedente concluir el presente capítulo ofreciendo una comparativa entre los resultados proporcionados por el criterio *SAD* y los obtenidos mediante la

aplicación del criterio de *Hamming*, la cual se muestra gráficamente a continuación en la *figura 3.2.18*:



(a) (b)

Figura 3.2.18. Gráficas comparativas de la tasa de acierto (a) y tiempo de ejecución (b) para la imagen *Cones*, según los algoritmos *SAD*, *SAD-Rank* y *Census Disperso*.

La figura muestra la tasa de acierto y el tiempo de ejecución obtenidos por los algoritmos *SAD*, *SAD* con transformada *Rank* y *Census Disperso*. En el caso del algoritmo *Census Disperso* los resultados mostrados son los proporcionados con el patrón de 2 píxeles cuando se aplica sobre la ventana de coste para un tamaño de ventana de transformación de *7x7*, así como también los resultados mostrados para *SAD-Rank* son los obtenidos a partir de una ventana de transformación de *7x7*. El tiempo de ejecución mostrado para *SAD y SAD-Rank* es el obtenido mediante el empleo de la técnica de reutilización de valores.

Analizando los resultados se podría concluir en que parece más conveniente hacer uso de los algoritmos basados en el criterio *SAD*, puesto que proporcionan una tasa de acierto tan buena o incluso superior a la proporcionada por los algoritmos basados en el criterio de *Hamming*, teniendo en cuenta que su coste computacional es mucho menor y por lo tanto proporcionan un tiempo de ejecución más aceptable.

CAPÍTULO 4. RESULTADOS OBTENIDOS PARA LAS IMÁGENES REALES Y ESPECÍFICAS

Hasta ahora, se ha llevado a cabo un meticuloso análisis acerca de la eficacia de los algoritmos para la obtención de mapas de disparidad presentados en el *capítulo 2* del presente proyecto, a través de su aplicación sobre una serie de imágenes pertenecientes a la *Universidad de Middlebury* diseñadas específicamente para este propósito. Sin embargo, resulta imprescindible realizar esa misma comparativa sobre imágenes reales, si bien pudiera haber diferencias o matizaciones sobre los resultados obtenidos en el capítulo anterior, pues no dejan de ser de índole teórica, al estar basados en la aplicación sobre imágenes prediseñadas. Dicha comparativa es el objeto del presente capítulo.

Para ello, en primer lugar, se ha llevado a cabo la captura de imágenes reales y específicas empleando un sistema *webcam* estéreo, mostrado en la siguiente figura:

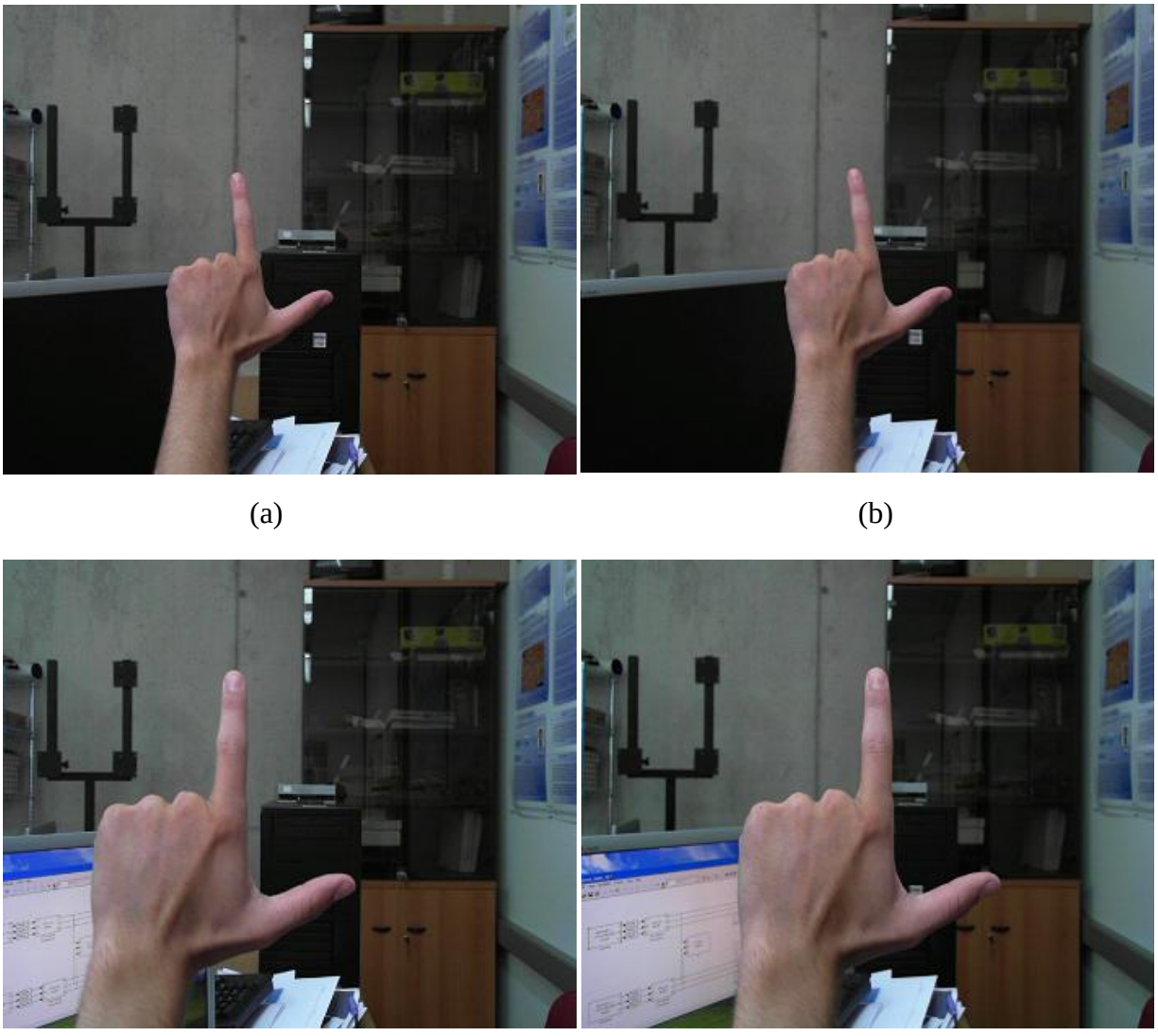
Figura 4.1. Sistema *webcam* estéreo empleado para la obtención de las imágenes reales y específicas.

Dicho sistema consiste en un par de *webcams* idénticas situadas paralelamente sobre un raíl que permite controlar de forma muy práctica la distancia de separación entre ambas cámaras. Además, el sistema está montado de tal forma que los ejes horizontales del par estéreo estén alineados con la mayor precisión posible, procurando así que sus líneas epipolares sean coincidentes, evitando de esta manera tener que realizar una posterior rectificación de las imágenes capturadas antes de llevar a cabo la aplicación de los algoritmos sobre las mismas.

El procedimiento para la captura de las imágenes reales y específicas mediante el sistema mostrado ha sido realizado en uno de los laboratorios de la *Universidad Politécnica de Cartagena*, mediante la ayuda de un *script* en *Matlab* para efectuar el

disparo de ambas cámaras de forma simultánea, y habiendo considerado la correspondiente calibración de las mismas.

A continuación, la *figura 4.2* muestra los dos pares estéreo de imágenes reales y específicas para las que se mostrarán en el presente capítulo los resultados obtenidos por cada uno de los algoritmos:



(a) (b)

(c) (d)

Figura 4.2. Imágenes par 1 izquierda (a), par 1 derecha (b), par 2 izquierda (c) y par 2 derecha (d).

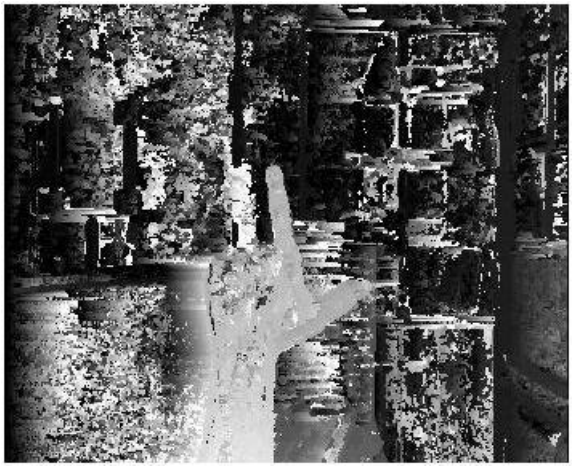
Es importante notar que, en este caso, no se cuenta con un mapa de disparidad ideal para cada uno de estos pares estéreo con el que realizar una valoración precisa acerca de los resultados obtenidos por los algoritmos, tal y como se hizo en el capítulo anterior donde se podía obtener la tasa de acierto alcanzada con cada uno de ellos. No obstante, teniendo en cuenta los objetivos del presente proyecto, será suficiente con

analizar visualmente los mapas de disparidad proporcionados, ya que la única pretensión es la de comprobar que el objeto situado en primer plano en la imagen queda claramente marcado en el mapa, y, como añadido, se mostrará la comparación visual del resultado obtenido con el conseguido mediante el empleo de las funciones proporcionadas por *Matlab* destinadas a la obtención de mapas de disparidad a partir de imágenes estereoscópicas.

La resolución de las imágenes mostradas en la *figura 4.2* sobre las que se han aplicado los algoritmos es de *352x288*, con lo cual los resultados referentes a los tiempos de ejecución (aunque no son objeto de este proyecto), son muy similares a los mostrados en el *capítulo 3* para el caso de la imagen *Tsukuba* de la *Universidad de Middlebury*. A continuación, se mostrarán en primer lugar los resultados obtenidos por los algoritmos basados en el criterio *SAD*, para posteriormente analizar los conseguidos mediante los algoritmos basados en el criterio *Hamming* y mostrar una comparativa final.

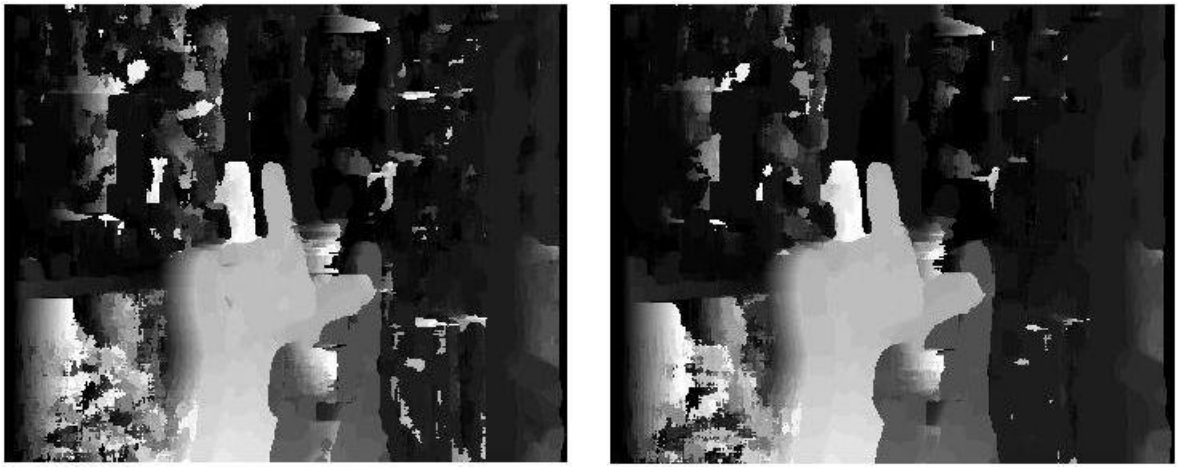
4.1 Resultados obtenidos para los algoritmos basados en *SAD*

A continuación, la *figura 4.1.1* y la *figura 4.1.2* muestran, respectivamente, los mapas de disparidad obtenidos al aplicar el algoritmo *SAD* básico sobre los pares de imágenes reales 1 y 2:



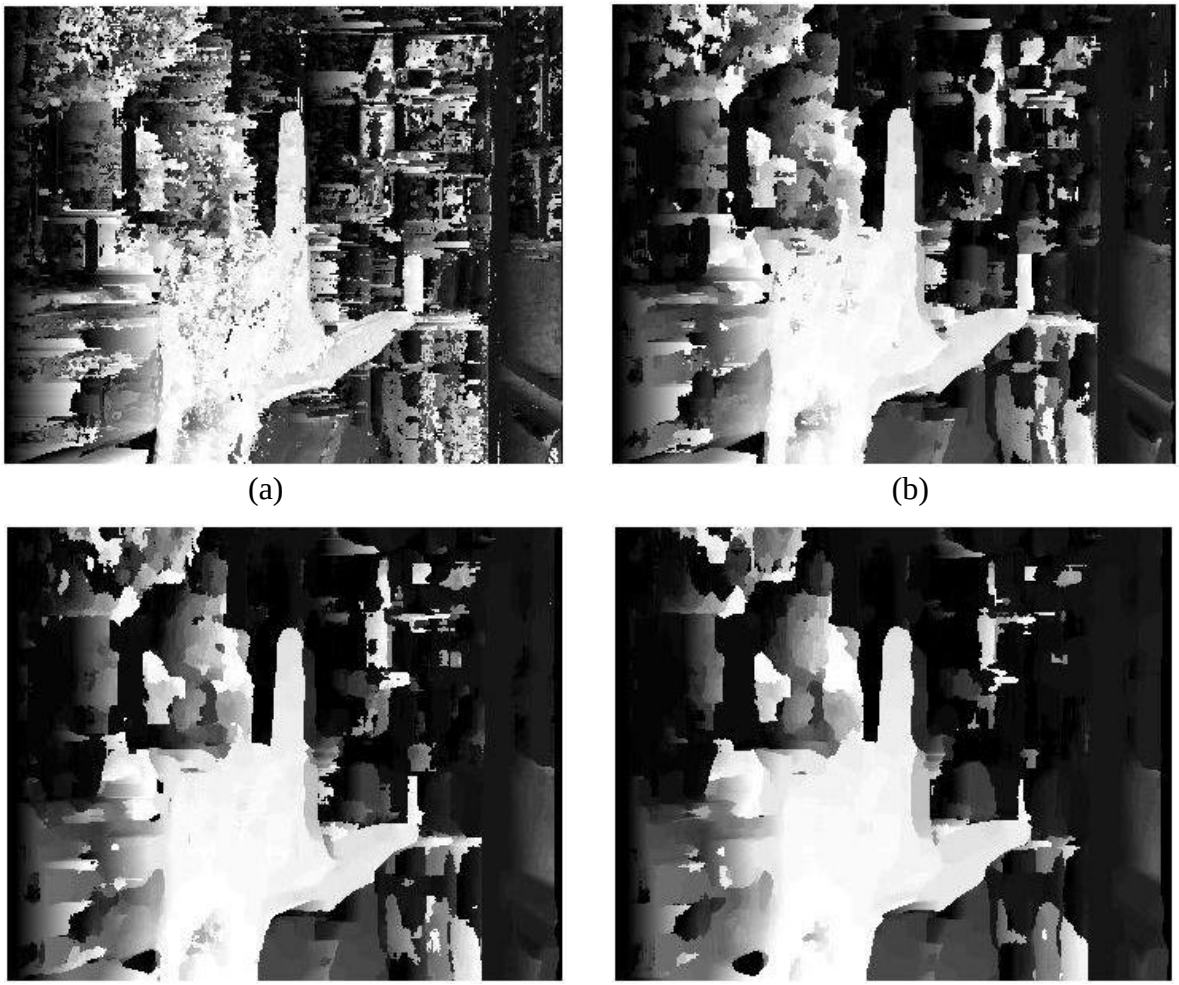
(a)

(b)



(c) (d)

Figura 4.1.1. Mapas de disparidad al aplicar *SAD* sobre el par 1 para tamaños de ventana de coste *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d).



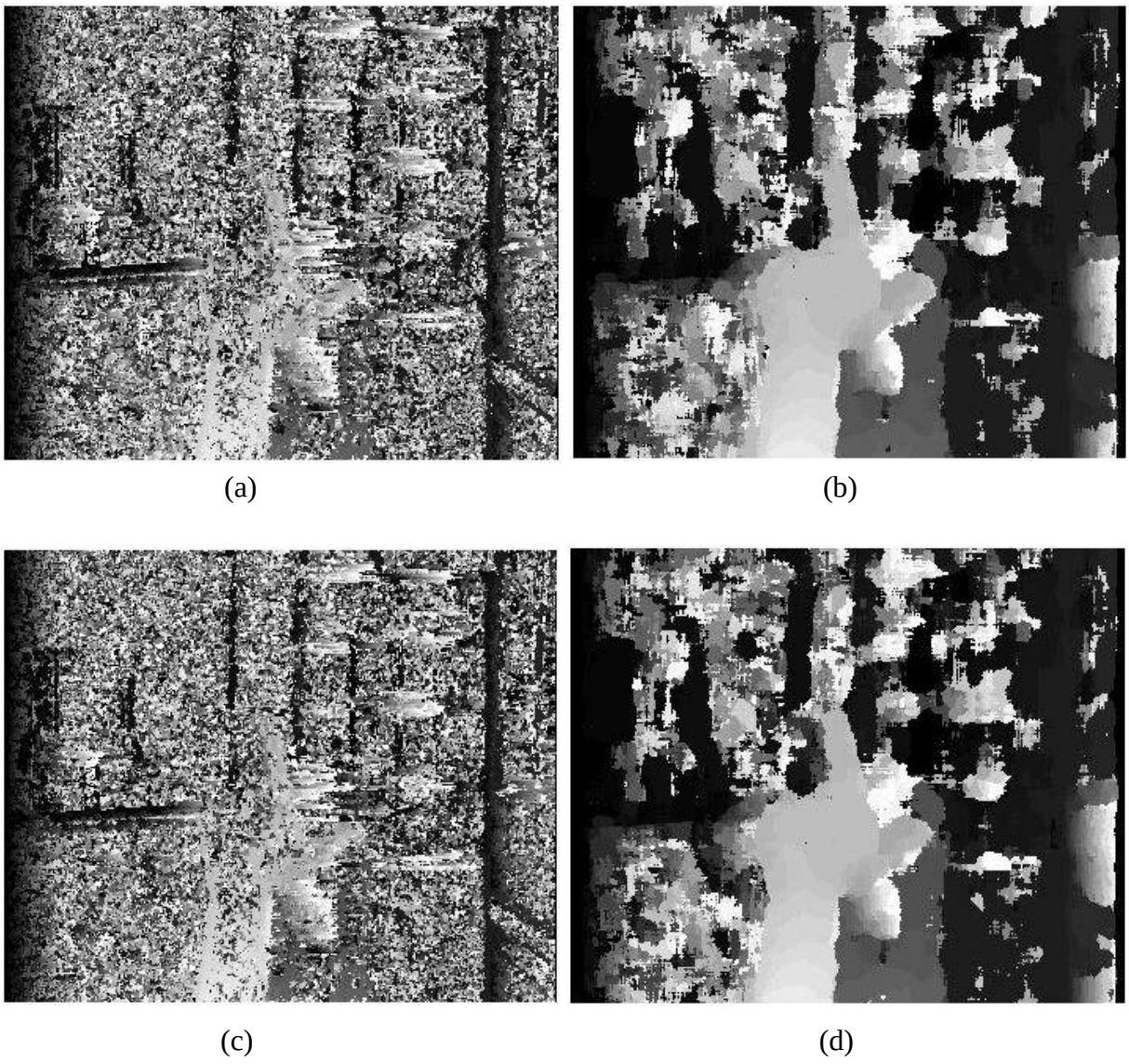
(a) (b)

(c) (d)

Figura 4.1.2. Mapas de disparidad al aplicar *SAD* sobre el par 2 para tamaños de ventana de coste *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d).

Al analizar los resultados de las pruebas realizadas sobre las imágenes de *Middlebury* en el capítulo anterior para el algoritmo *SAD* básico, se observó que conforme se aumentaba el tamaño de la ventana de coste, la tasa de acierto conseguida también aumentaba. Analizando ahora los resultados mostrados en las *figuras 4.1.1* y *4.1.2* para las imágenes reales se puede confirmar ese hecho, pues se aprecia cómo, al aumentar la ventana de coste empleada por el algoritmo, el ruido se ve reducido y el área de la imagen delimitada por la mano (que corresponde al objeto que está en primer plano) queda mejor definida.

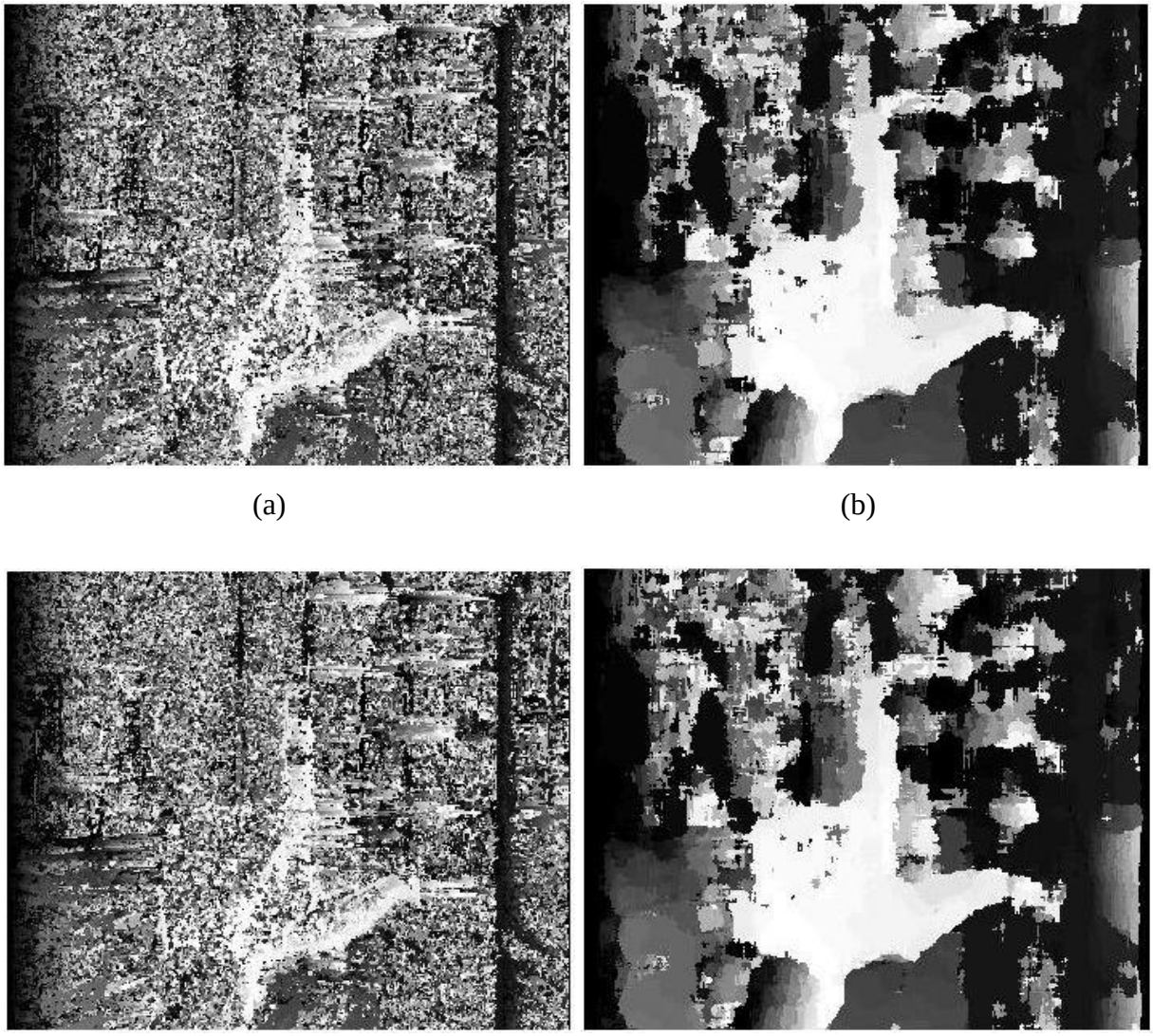
Este hecho se confirma también para los algoritmos *Rank* y *Soft-Rank*, tal y como puede apreciarse a continuación en las *figuras 4.1.3* y *4.1.4*:



(a) (b)

(c) (d)

Figura 4.1.3. Mapas de disparidad al aplicar *Rank* para tamaños de ventana de coste *3x3* (a) y *15x15* (b), y *Soft-Rank* para tamaños de ventana de coste *3x3* (c) y *15x15* (d), sobre el par 1.



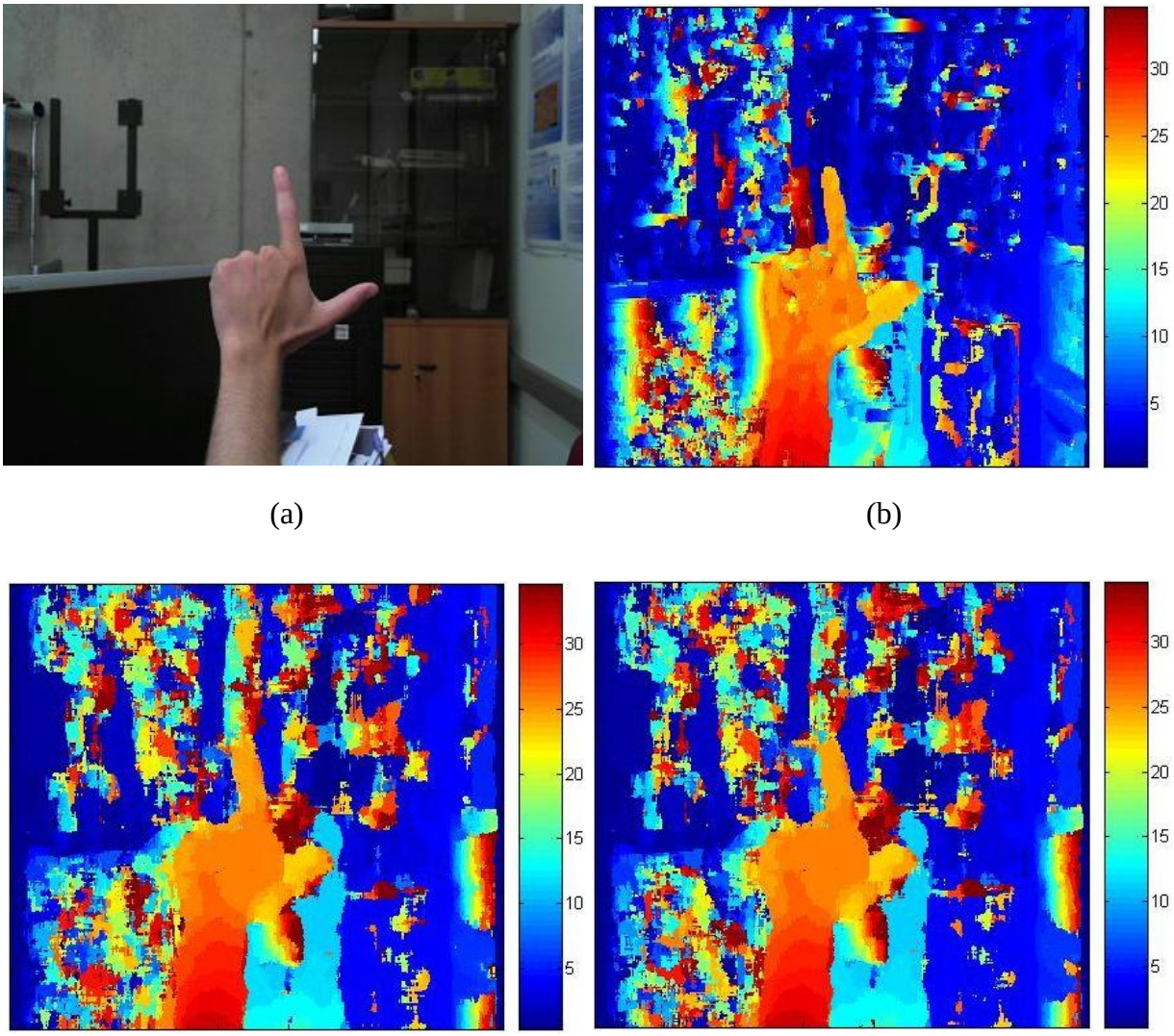
(a) (b)

(c) (d)

Figura 4.1.4. Mapas de disparidad al aplicar *Rank* para tamaños de ventana de coste *3x3* (a) y *15x15* (b), y *Soft-Rank* para tamaños de ventana de coste *3x3* (c) y *15x15* (d), sobre el par 2.

De nuevo, observando los resultados obtenidos (en los que se ha empleado un tamaño de ventana de transformación de *7x7* para ambos algoritmos y un *umbral Soft-Rank* de 2), se puede apreciar que al aumentar el tamaño de la ventana de coste tanto para el algoritmo *Rank* como para el algoritmo *Soft-Rank* aumenta de forma bastante marcada la calidad del mapa de disparidad obtenido, reduciéndose en gran medida el ruido y quedando mejor delimitada el área de la imagen correspondiente a la mano. No obstante, ninguno de ellos proporciona resultados mejores que los conseguidos mediante el algoritmo *SAD* básico, ni siquiera para un tamaño de ventana de coste de *15x15*. Esto confirma los resultados obtenidos analizados en el capítulo anterior para el caso particular de la imagen *Tsukuba*, cuya resolución coincide con la de las imágenes reales empleadas en el presente capítulo.

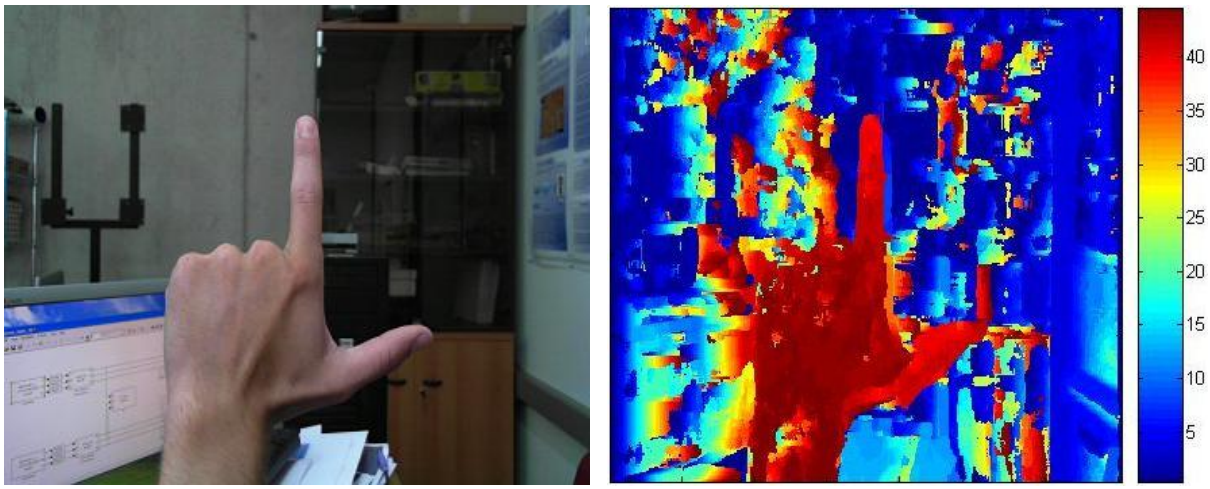
Mediante el empleo de la función *imagesc()* de *Matlab* se puede apreciar con mejor claridad de forma visual la calidad de los mapas de disparidad obtenidos por los algoritmos. Las figuras *4.1.5* y *4.1.6* muestran para el par 1 y el par 2 respectivamente, la comparación entre los resultados proporcionados por *SAD*, *Rank* y *Soft-Rank*. El parámetro de máxima disparidad que se ha empleado para las pruebas es de *35* para el par 1 y de *45* para el par 2, tal y como muestran las *colorbars* añadidas en las imágenes mostradas.



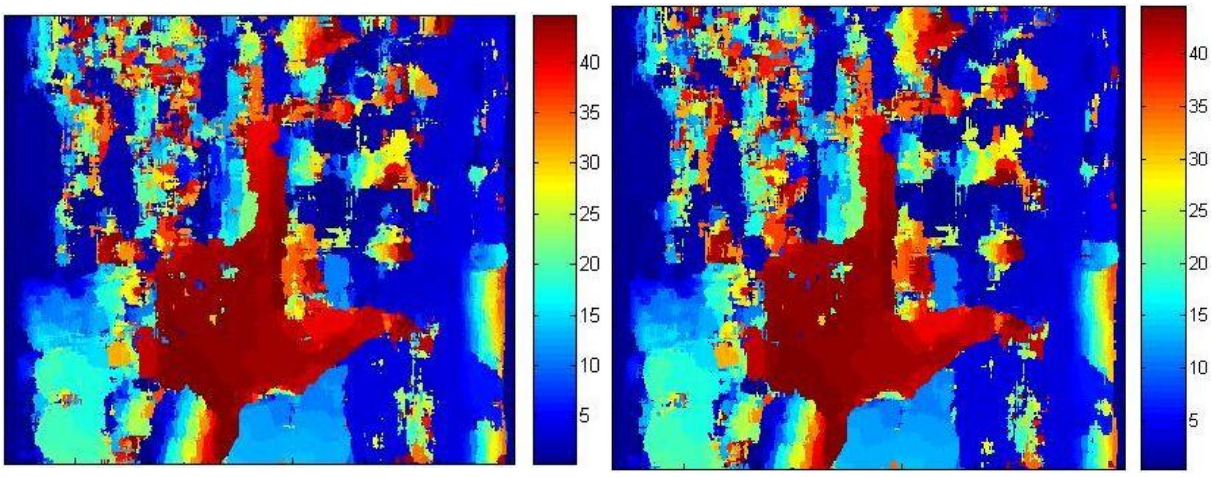
(a) (b)

(c) (d)

Figura 4.1.5. Mapas de disparidad al aplicar *SAD* (b), *Rank* (c) y *Soft-Rank* (d) sobre el par 1.



(a) (b)



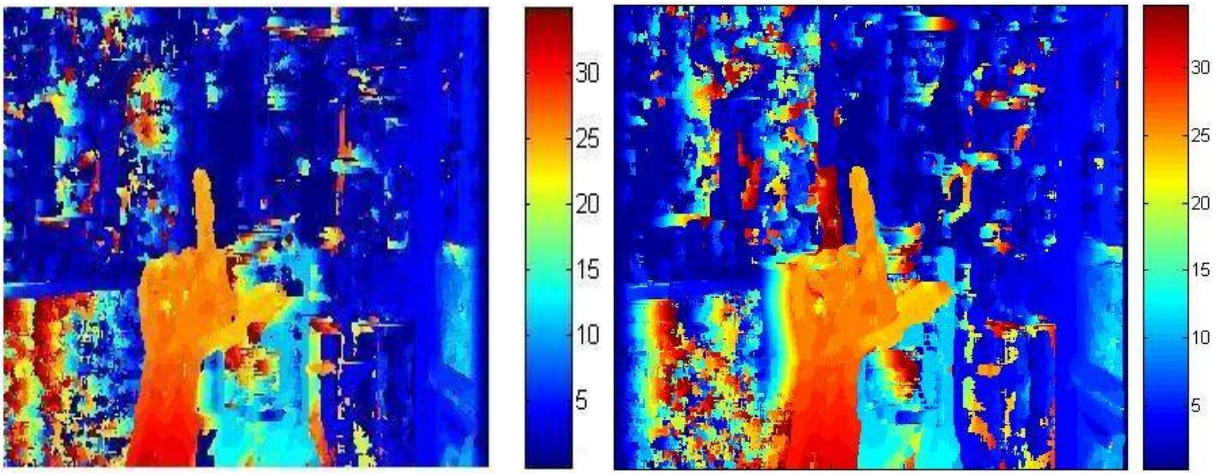
(c) (d)

Figura 4.1.6. Mapas de disparidad al aplicar *SAD* (b), *Rank* (c) y *Soft-Rank* (d) sobre el par 2.

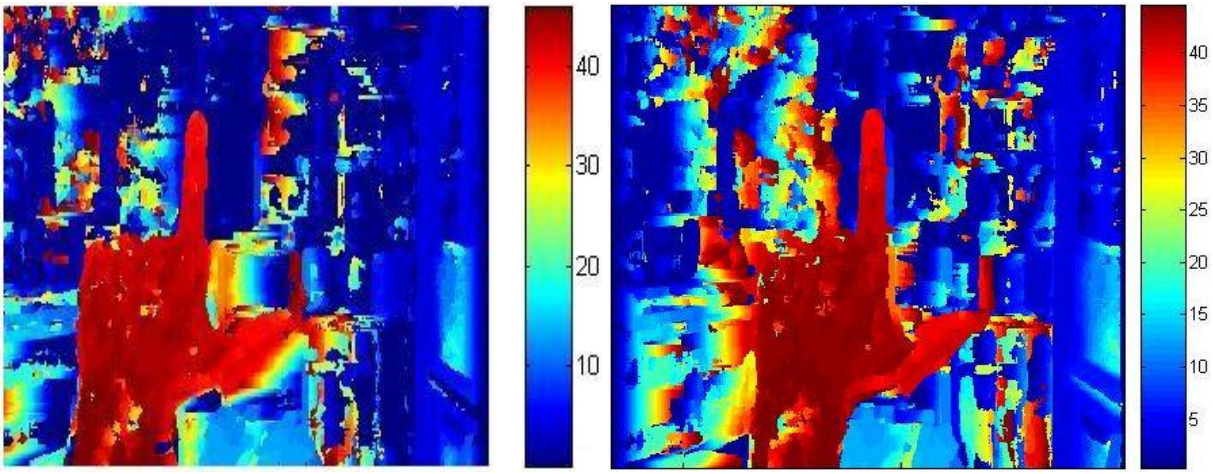
En el caso del algoritmo *SAD* se han seleccionado los mapas de disparidad proporcionados para un tamaño de ventana de coste de *7x7*, mientras que para los algoritmos *Rank* y *Soft-Rank* se han seleccionado los mapas de disparidad obtenidos con una ventana de coste de *15x15*. Se puede apreciar fácilmente que los resultados conseguidos con *Rank* y con *Soft-Rank* son casi idénticos entre sí, y además no son mejores que los resultados obtenidos mediante el algoritmo *SAD* básico incluso a pesar de estar empleando en la aplicación de dicho algoritmo un tamaño de ventana de coste mucho menor que el empleado con *Rank* y *Soft-Rank*. Estos datos confirman por tanto las conclusiones alcanzadas tras el análisis de las pruebas detalladas en el capítulo anterior para el caso de los algoritmos basados en el criterio *SAD*.

A continuación, la *figura 4.1.7* muestra para los pares de imágenes reales una comparación entre los mapas de disparidad obtenidos mediante las funciones propias de

Matlab y los obtenidos mediante la aplicación del algoritmo *SAD* básico implementado para el presente proyecto:



(a) (b)



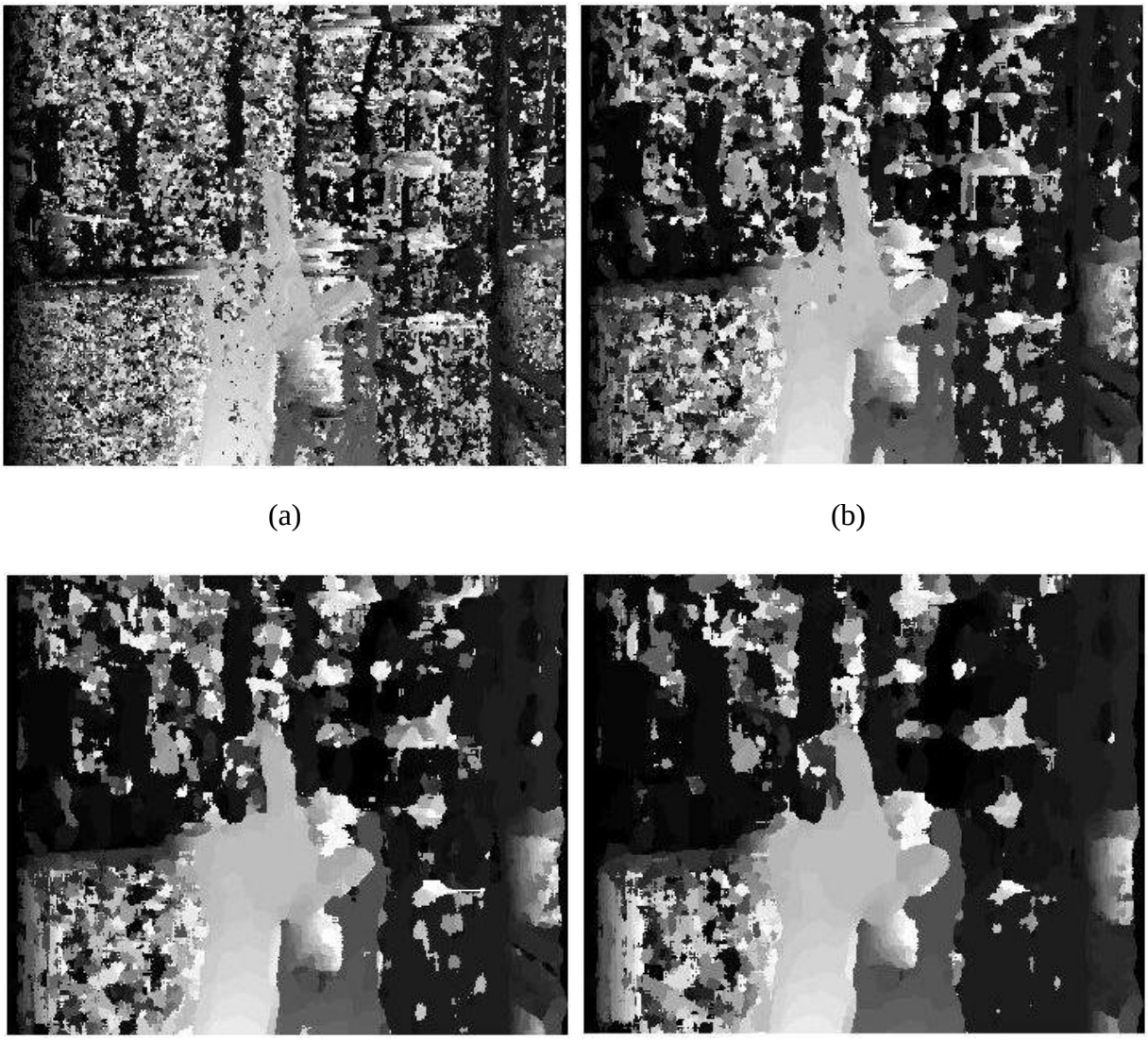
(c) (d)

Figura 4.1.7. Mapas de disparidad obtenidos por *Matlab* para el par 1 (a) y para el par 2 (c), y por *SAD* básico para el par 1 (b) y el par 2 (d).

Los mapas de disparidad seleccionados para el caso de *SAD* son los obtenidos mediante una ventana de coste de tamaño *7x7*. Se puede apreciar que los resultados obtenidos son muy similares entre sí. Aunque quizás el resultado proporcionado por *Matlab* sea algo mejor, la diferencia no es demasiado marcada, y empleando tamaños de ventana mayores se puede mejorar. No obstante, se trata de un buen resultado, ya que no difiere demasiado del proporcionado por el propio *Matlab* (lo cual es un buen punto de comparación), y se distingue claramente el área en primer plano delimitada por la mano como objeto principal de las escenas.

4.2 Resultados obtenidos para los algoritmos basados en *HAMMING*

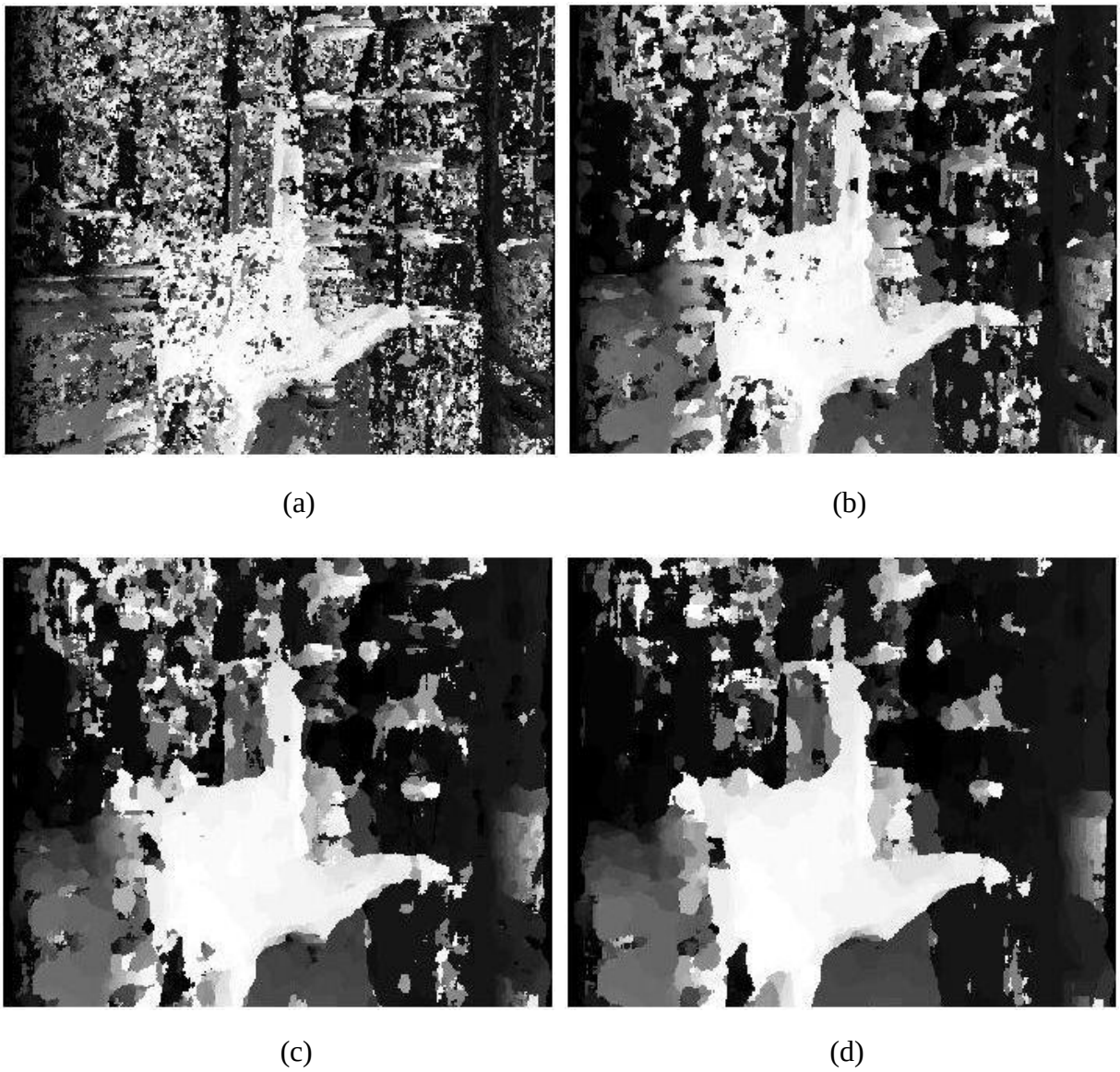
A continuación, la *figura 4.2.1* y la *figura 4.2.2* muestran, respectivamente, los mapas de disparidad obtenidos al aplicar el algoritmo *Census* en su versión original sobre los pares de imágenes reales 1 y 2:



(a) (b)

(c) (d)

Figura 4.2.1. Mapas de disparidad al aplicar *Census* sobre el par 1 para tamaños de ventana de coste *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d).



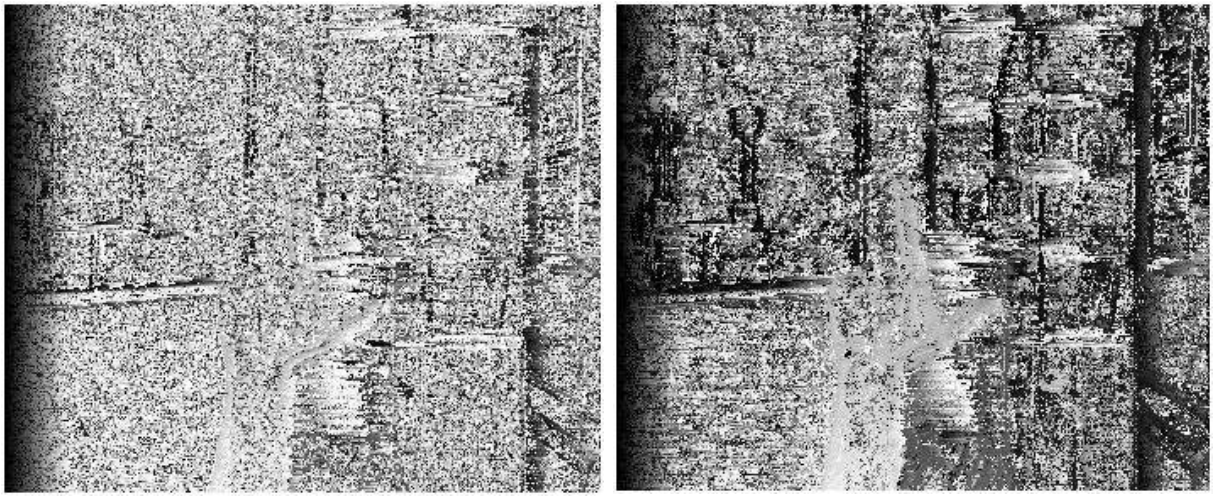
(a) (b)

(c) (d)

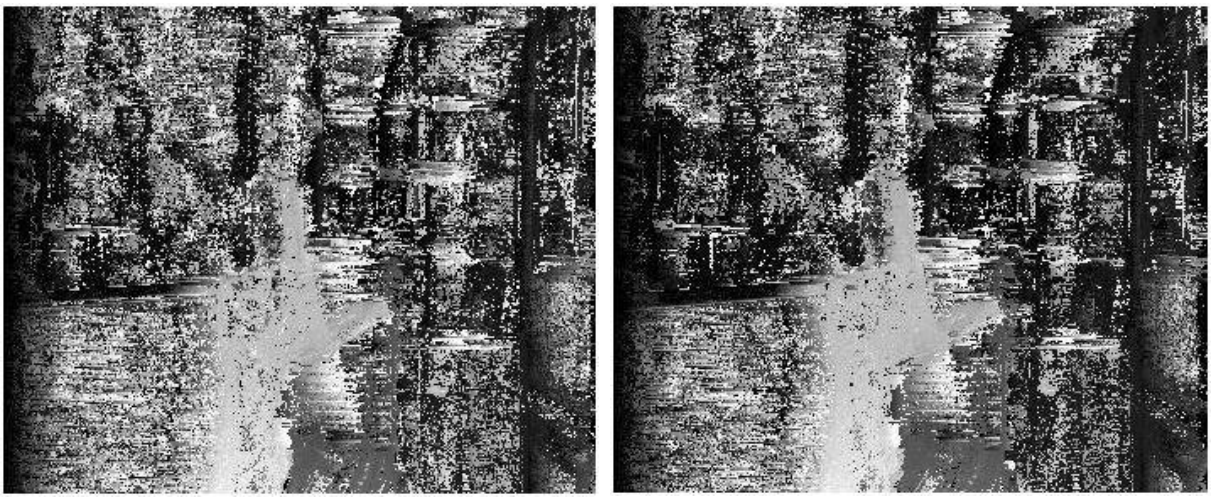
Figura 4.2.2. Mapas de disparidad al aplicar *Census* sobre el par 2 para tamaños de ventana de coste *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d).

Las pruebas han sido realizadas para un tamaño de ventana de transformación de *7x7*. Es fácil observar que, al aumentar el tamaño de la ventana de coste, el ruido presente en el mapa es cada vez menor, pero, por otra parte, también se ve afectada la definición del objeto situado en primer plano. Al pasar de un tamaño de ventana de coste de *7x7*, se va apreciando cada vez una deformación más clara del área correspondiente a la mano, situada en primer plano en la escena.

En el caso del algoritmo *mini-Census*, los resultados son los mostrados a continuación en las *figuras 4.2.3* y *4.2.4*:

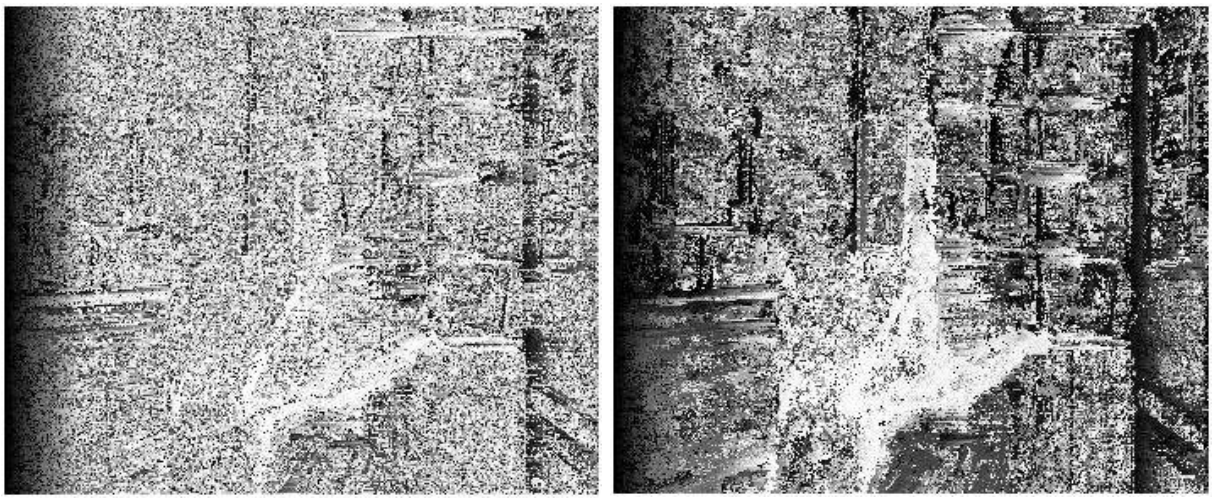


(a) (b)

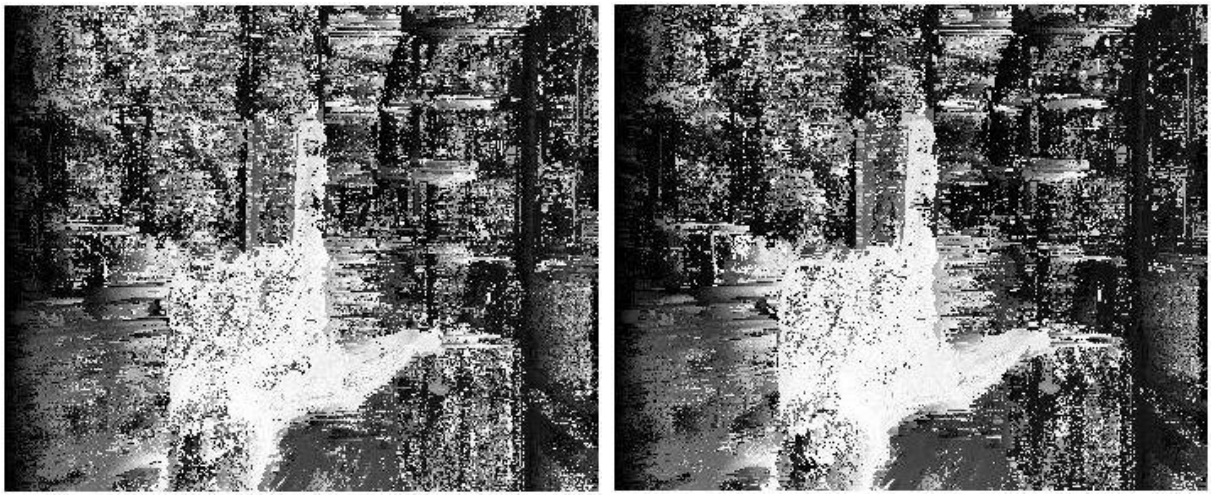


(c) (d)

Figura 4.2.3. Mapas de disparidad al aplicar *mini-Census* sobre el par 1 para tamaños de ventana de transformación *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d).



(a) (b)



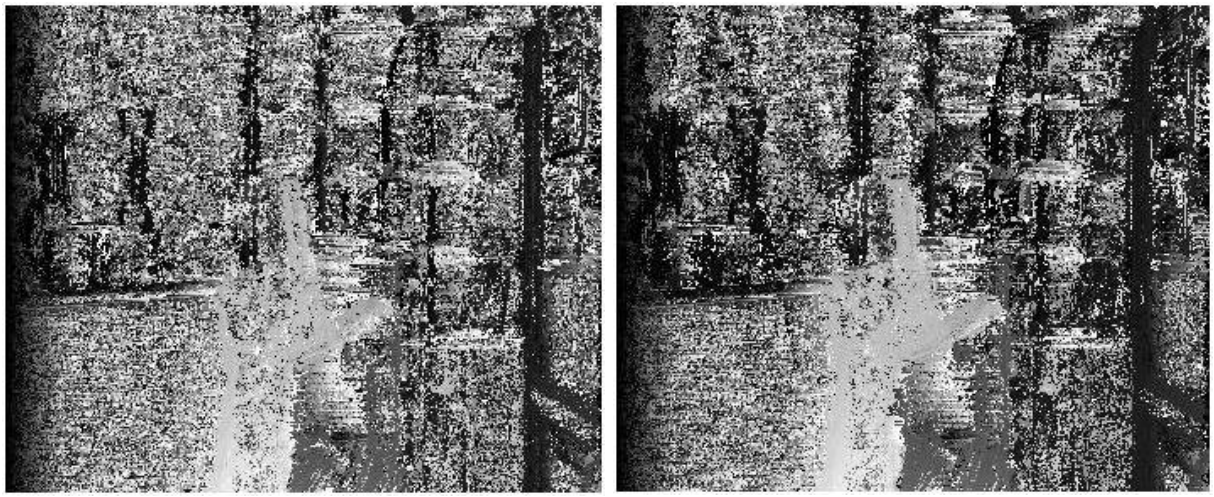
(c) (d)

Figura 4.2.4. Mapas de disparidad al aplicar *mini-Census* sobre el par 2 para tamaños de ventana de transformación *3x3* (a), *7x7* (b), *11x11* (c) y *15x15* (d).

Se puede observar en las imágenes que, al aumentar el tamaño de la ventana de transformación, aumenta también la calidad del mapa obtenido reduciéndose el ruido presente en el mismo y pudiendo apreciar cada vez un poco mejor el área perteneciente a la mano situada en primer plano en las escenas. No obstante esta reducción de ruido se produce de forma menos abrupta que en el caso del algoritmo *Census* original, y se aprecia más entre tamaños de ventana reducidos (como al pasar del tamaño de *3x3* al de *7x7*) que entre tamaños de ventana más amplios (como al pasar del tamaño de *11x11* al de *15x15*), lo cual confirma los resultados que se obtuvieron para este algoritmo en el capítulo anterior para las imágenes de *Middlebury*.

En lo que respecta al algoritmo de *mini-Census Extendido*, en la *figura 4.2.5* se muestran los resultados obtenidos para ambos pares de imágenes reales empleando tamaños de ventana de transformación de *7x7* y *11x11*.

De nuevo, en este caso, puede apreciarse una cierta mejora en la calidad de los mapas obtenidos al aumentar el tamaño de la ventana de transformación, si bien es cierto que esta mejora no es demasiado marcada y los resultados producidos no son notablemente mejores que los obtenidos mediante el algoritmo *mini-Census*.



(a) (b)

(c) (d)

Figura 4.2.5. Mapas de disparidad al aplicar *mini-Census Extendido* sobre el par 1 para tamaños de ventana de transformación de *7x7* (a) y *11x11* (b), y sobre el par 2 para tamaños de ventana de transformación de *7x7* (c) y *11x11* (d).

Con la ayuda de la función *imagesc* de *Matlab* se puede apreciar mejor la calidad de los resultados obtenidos por los algoritmos. Las *figuras 4.2.6* y *4.2.7* muestran una comparativa entre los resultados proporcionados por los algoritmos *Census*, *mini-Census* y *mini-Census Extendido* para el par 1 y el par 2, respectivamente.

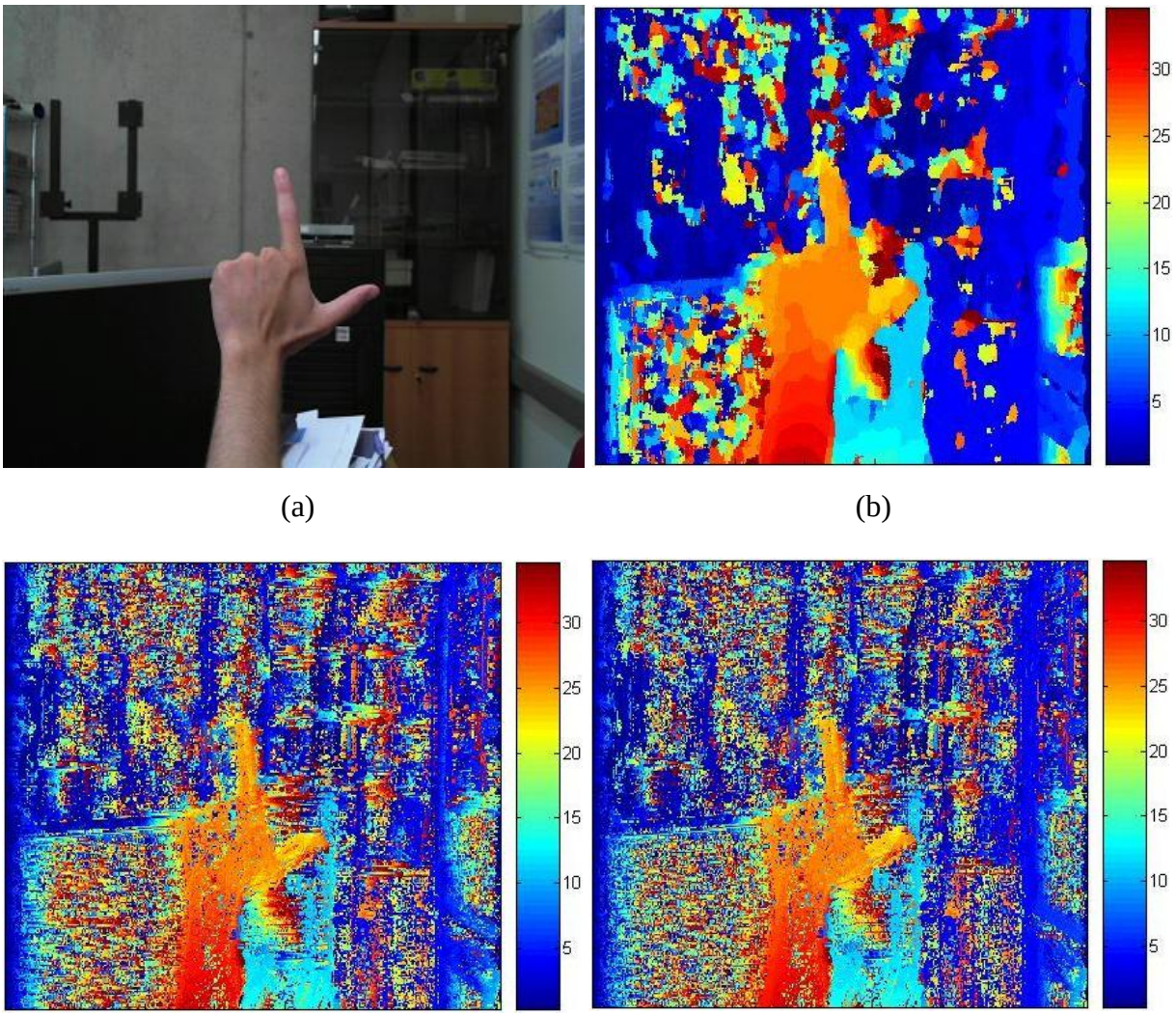
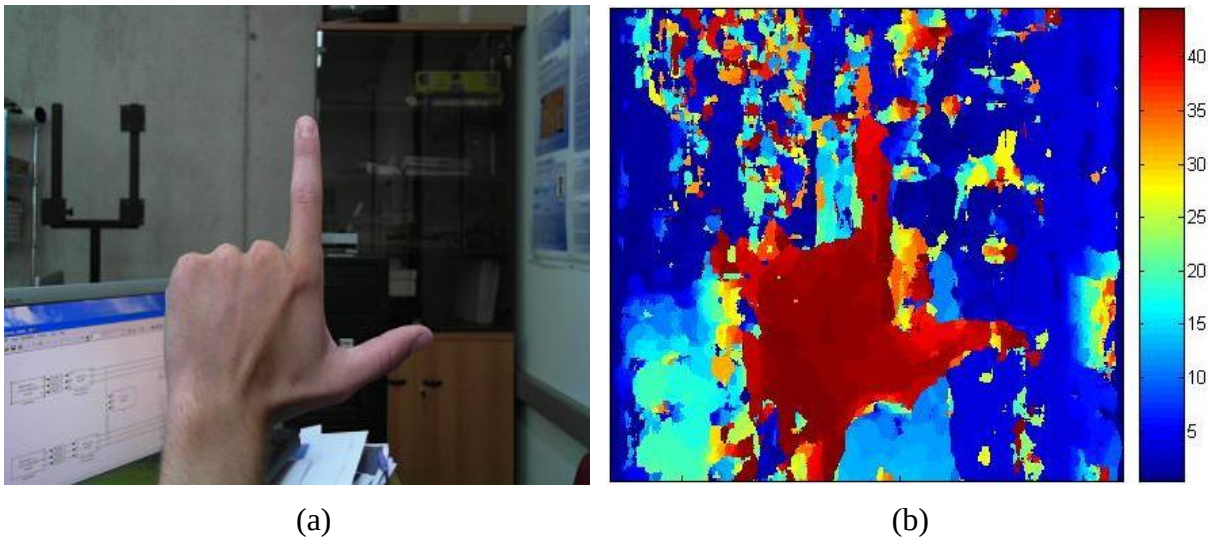
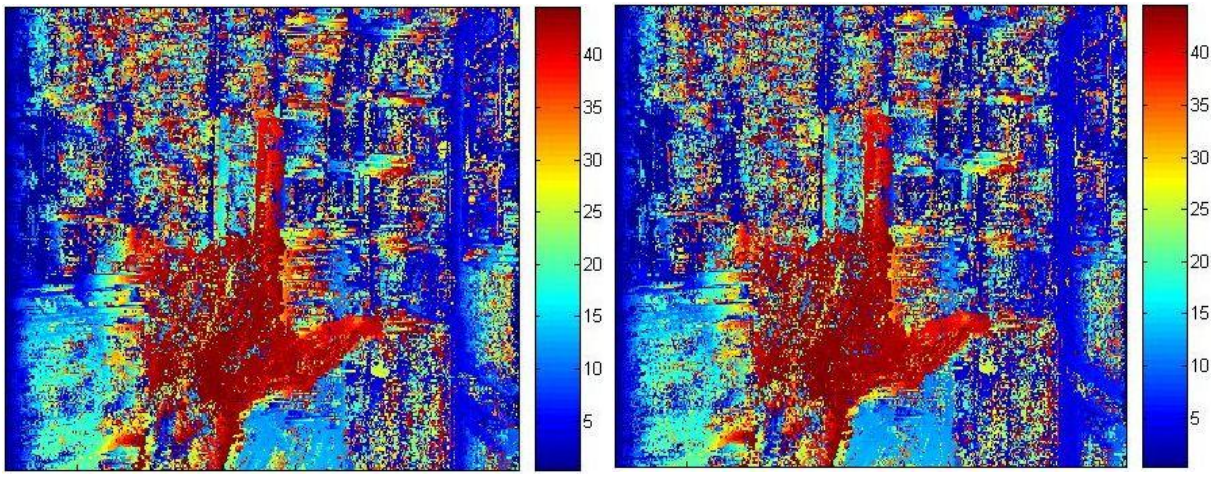


Figura 4.2.6. Mapas de disparidad al aplicar *Census* (b), *mini-Census* (c) y *mini-Census Extendido* (d) sobre el par 1.



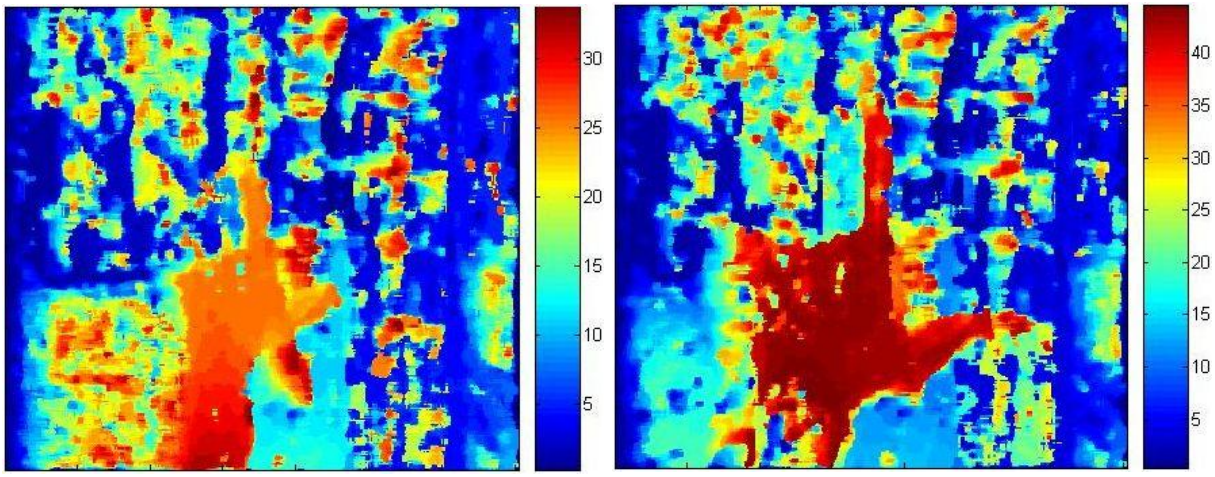


(c) (d)

Figura 4.2.7. Mapas de disparidad al aplicar *Census* (b), *mini-Census* (c) y *mini-Census Extendido* (d) sobre el par 2.

Los mapas mostrados corresponden a los obtenidos mediante el empleo de ventanas de trabajo de tamaño *11x11*. Se puede apreciar en las imágenes que el mejor resultado es el proporcionado por el algoritmo *Census* en su versión original, y que no hay apenas diferencia entre los mapas que resultan de aplicar los algoritmos *mini-Census* y *mini-Census Extendido*, con lo cual quedan confirmados los resultados que se obtuvieron para las imágenes de *Middlebury* en el capítulo anterior del presente proyecto.

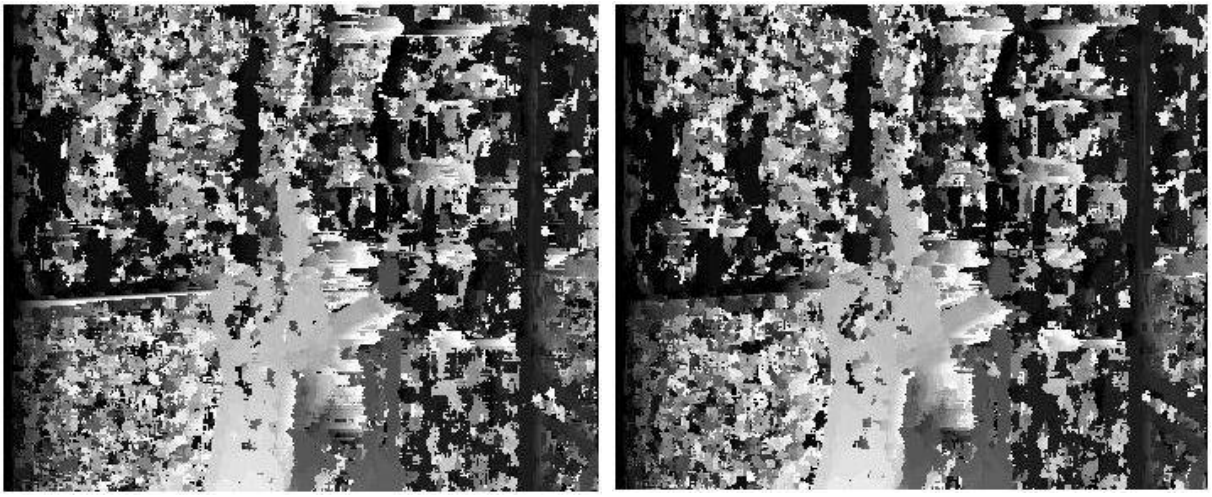
Al aplicar el filtro de la mediana como etapa de post-filtrado sobre los mapas de disparidad obtenidos mediante el algoritmo *mini-Census*, se puede notar cómo mejora la calidad del mapa, tal y como ilustra la *figura 4.2.8*, en la que se aprecian los resultados de aplicar el filtro sobre los mapas obtenidos para ambos pares de imágenes reales mediante el empleo de una ventana de transformación de tamaño *11x11* y un filtro de tamaño *7x7*, no obstante, a pesar de la notable mejoría en cuanto a reducción de ruido presente en el mapa y a la definición del área correspondiente e la mano, los resultados siguen sin ser mejores que los proporcionados por la aplicación del algoritmo *Census* en su versión original.



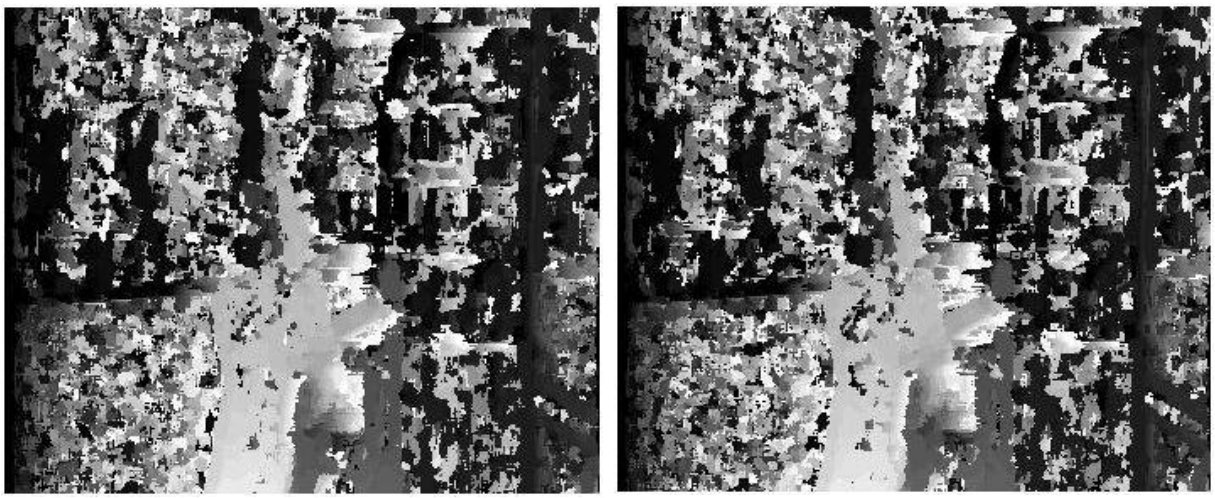
(a) (b)

Figura 4.2.8. Mapas de disparidad al aplicar un filtro de mediana de tamaño *7x7* tras emplear el algoritmo *mini-Census* con un tamaño de ventana de transformación de *11x11* para el par 1 (a) y el par 2 (b).

En lo que respecta al algoritmo *Census Disperso*, en primer lugar se procede a mostrar los resultados para el caso en el que el patrón de dispersión se aplica sobre la ventana de transformación, ilustrados en las *figuras 4.2.9 y 4.2.10* para cada uno de los pares de imágenes reales, respectivamente.

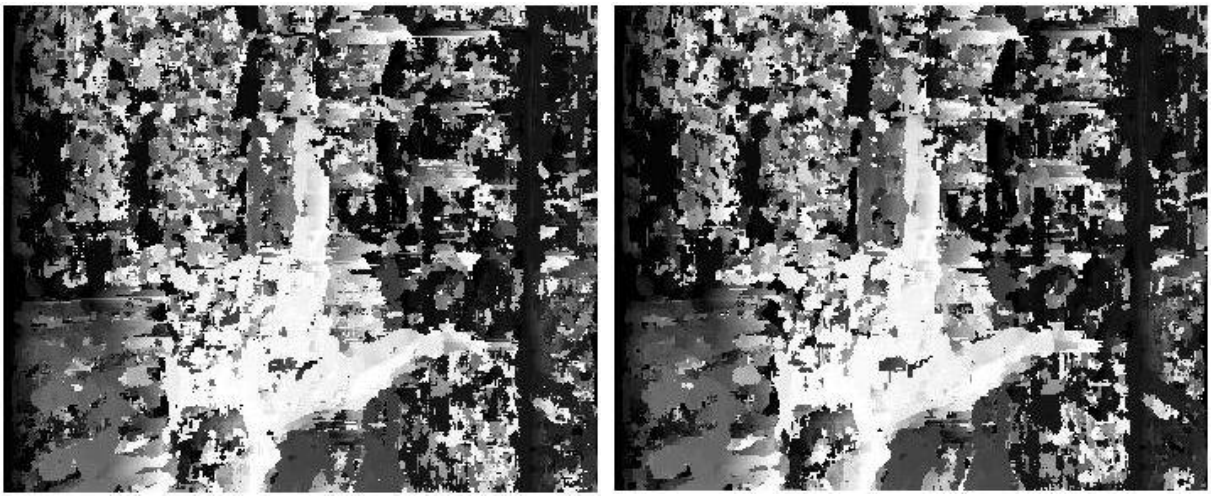


(a) (b)

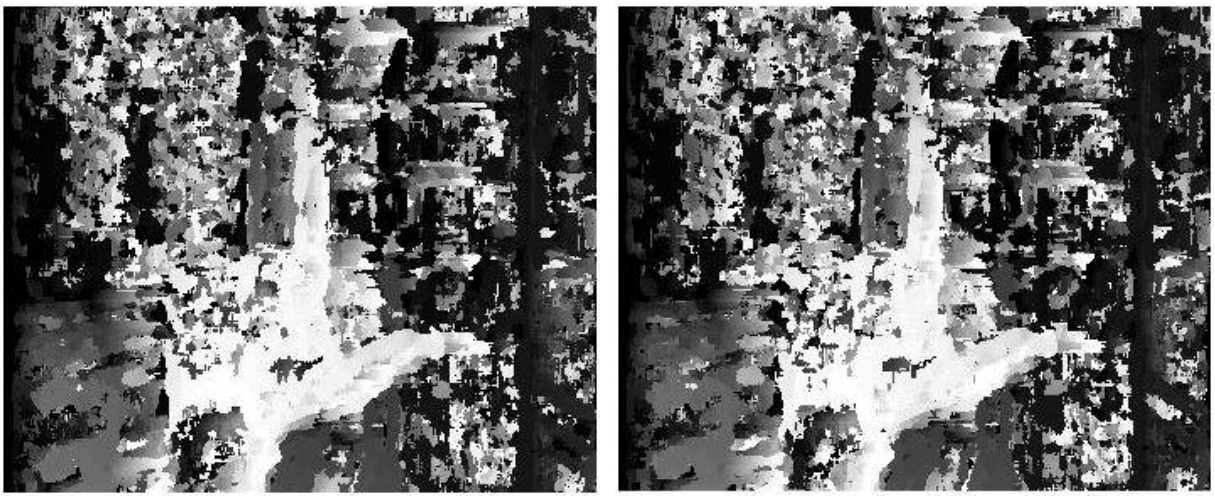


(c) (d)

Figura 4.2.9. Mapas de disparidad al aplicar *Census Disperso* sobre la ventana de transformación empleando un patrón de dispersión de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y en cruz (d), para el par 1.



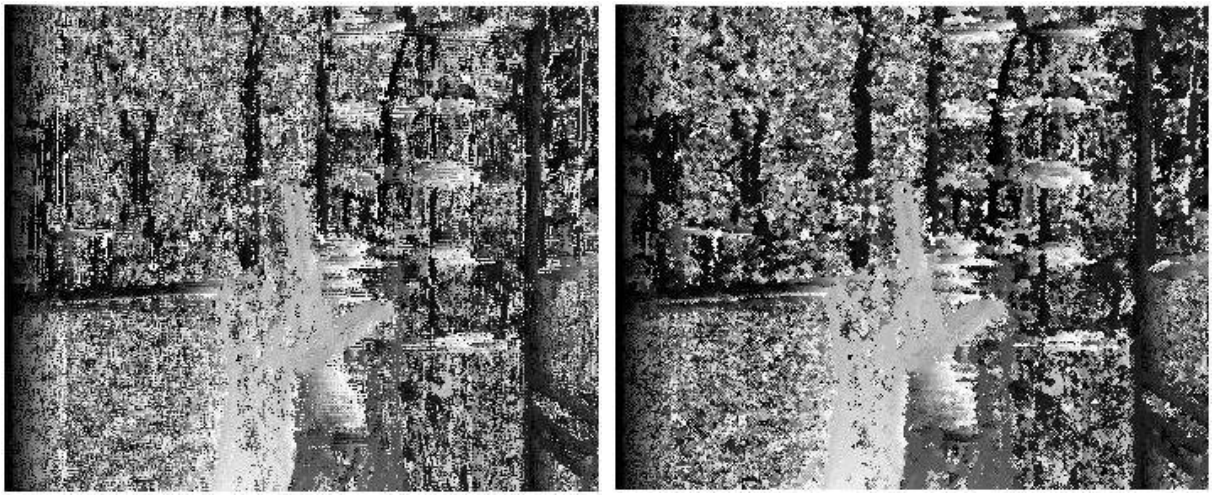
(a) (b)



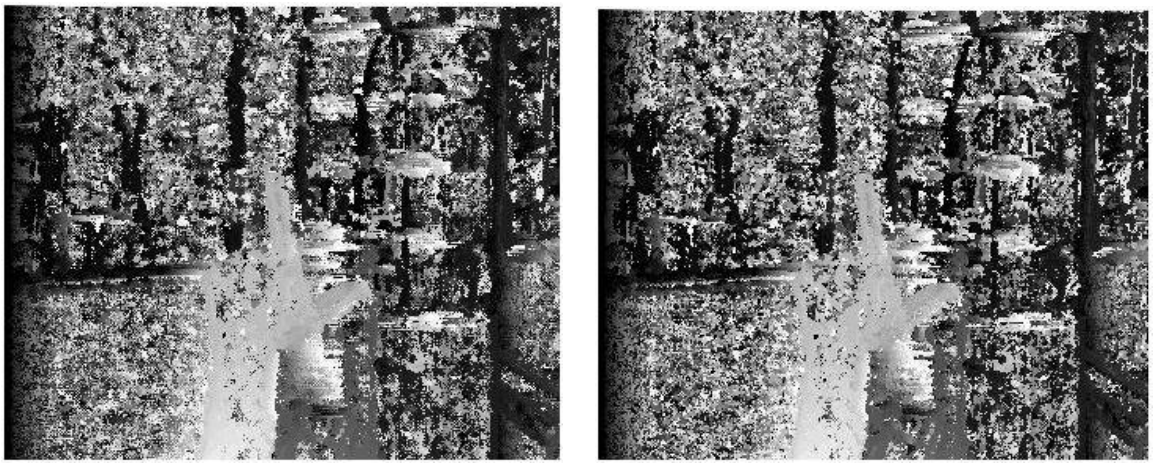
(c) (d)

Figura 4.2.10. Mapas de disparidad al aplicar *Census Disperso* sobre la ventana de transformación empleando un patrón de dispersión de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y en cruz (d), para el par 2.

Viendo los resultados para ambos pares de imágenes, se puede afirmar que apenas se aprecian diferencias al emplear un patrón u otro. Los mapas mostrados son los obtenidos mediante el empleo de una ventana de transformación de tamaño *7x7*. A continuación, las *figuras 4.2.11* y *4.2.12* muestran estos mismos resultados para el caso en el que el patrón de dispersión se aplica sobre la ventana de coste en lugar de sobre la ventana de transformación.

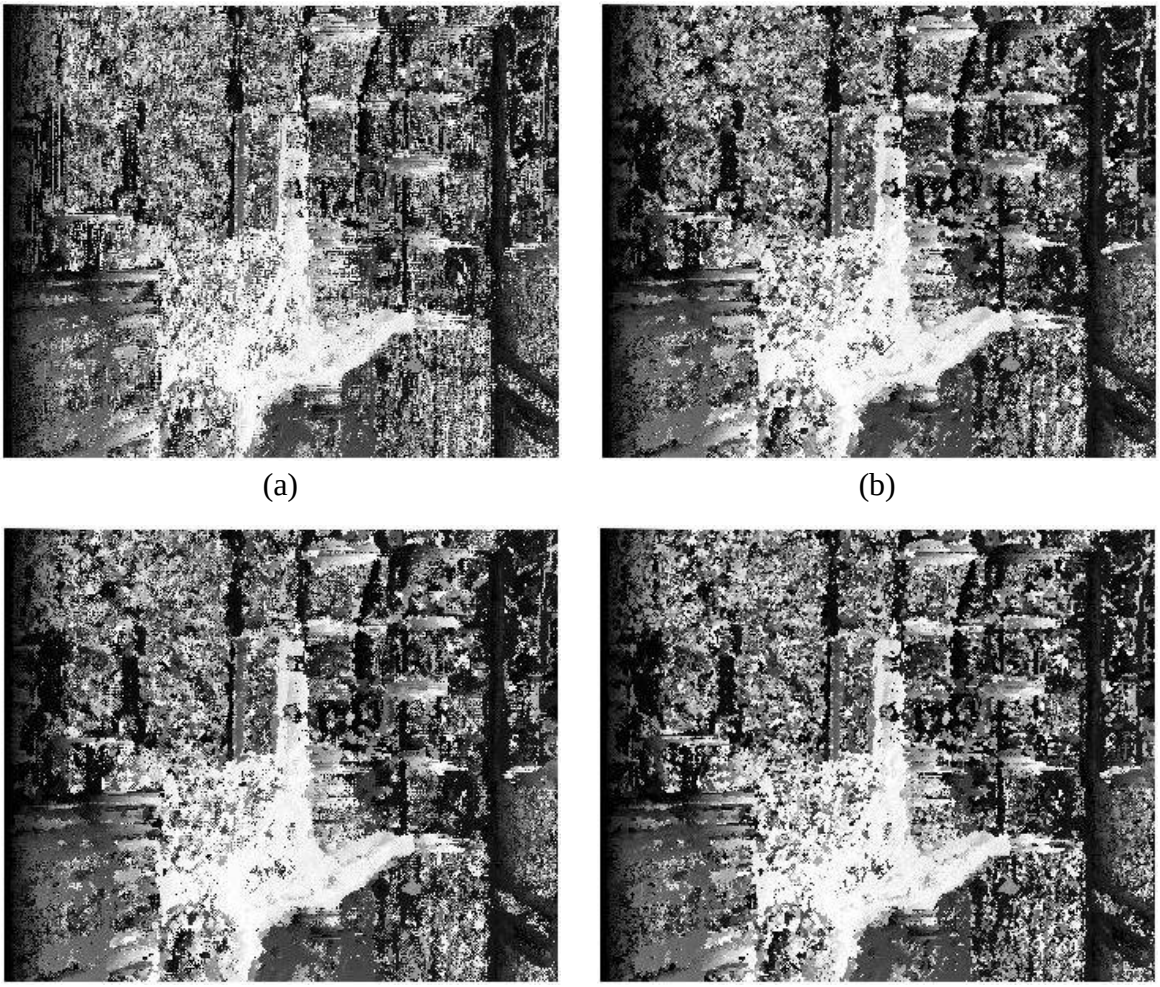


(a) (b)



(c) (d)

Figura 4.2.11. Mapas de disparidad al aplicar *Census Disperso* sobre la ventana de coste empleando un patrón de dispersión de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y en cruz (d), para el par 1.

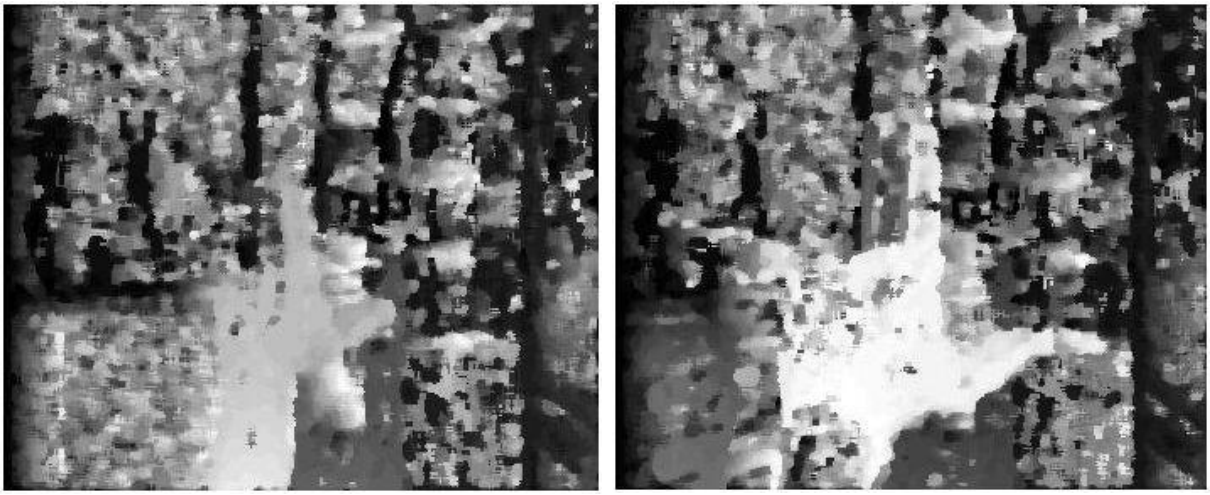


(a) (b)

(c) (d)

Figura 4.2.12. Mapas de disparidad al aplicar *Census Disperso* sobre la ventana de coste empleando un patrón de dispersión de 2 píxeles (a), 4 píxeles (b), 5 píxeles (c) y en cruz (d), para el par 2.

De nuevo, al igual que se ha comprobado para el caso en el que el patrón de dispersión se aplicaba sobre la ventana de transformación, se vuelve a observar también en este caso que el empleo de un patrón u otro apenas tiene repercusión sobre el resultado del mapa de disparidad obtenido. Esto es indicativo de que es factible emplear un patrón con el menor número de píxeles posible, como es el patrón de 2 píxeles, puesto que no hay apenas diferencia en el resultado con respecto a usar un patrón con un mayor número de píxeles, y se estaría reduciendo al mínimo el tiempo de ejecución demandado por el algoritmo. Sin embargo, los resultados obtenidos al aplicar el patrón de dispersión sobre la ventana de transformación, resultan ser algo mejores que al aplicar el patrón sobre la ventana de coste, puesto que se aprecia una menor presencia de ruido en los mapas correspondientes. Sería por tanto interesante observar los mapas resultantes para ambos pares de imágenes tras aplicar un filtro de mediana sobre los resultados proporcionados por el algoritmo cuando el patrón de 2 píxeles se aplica sobre la ventana de coste, los cuales se muestran a continuación en la *figura 4.2.13*:

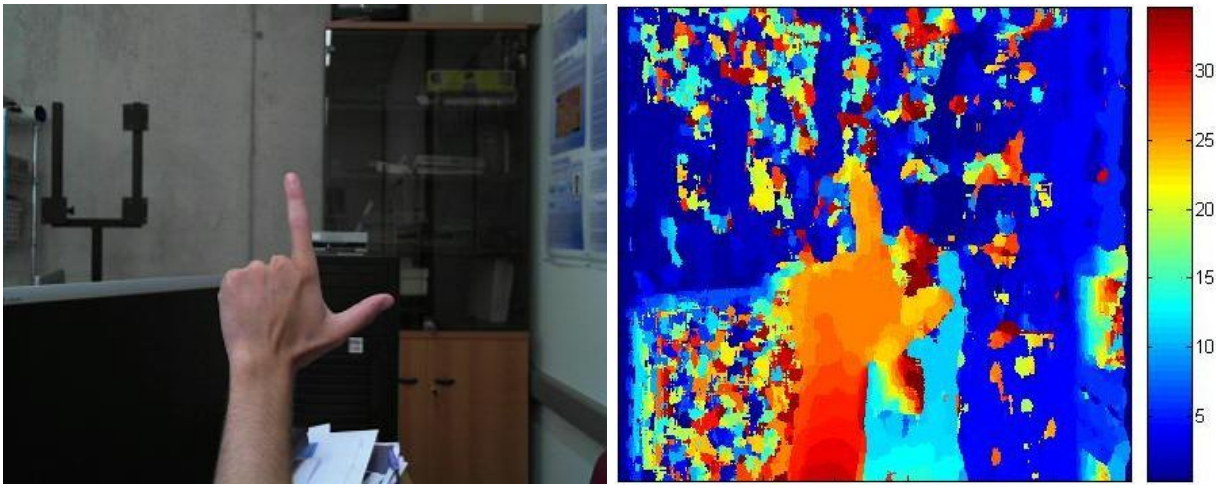


(a) (b)

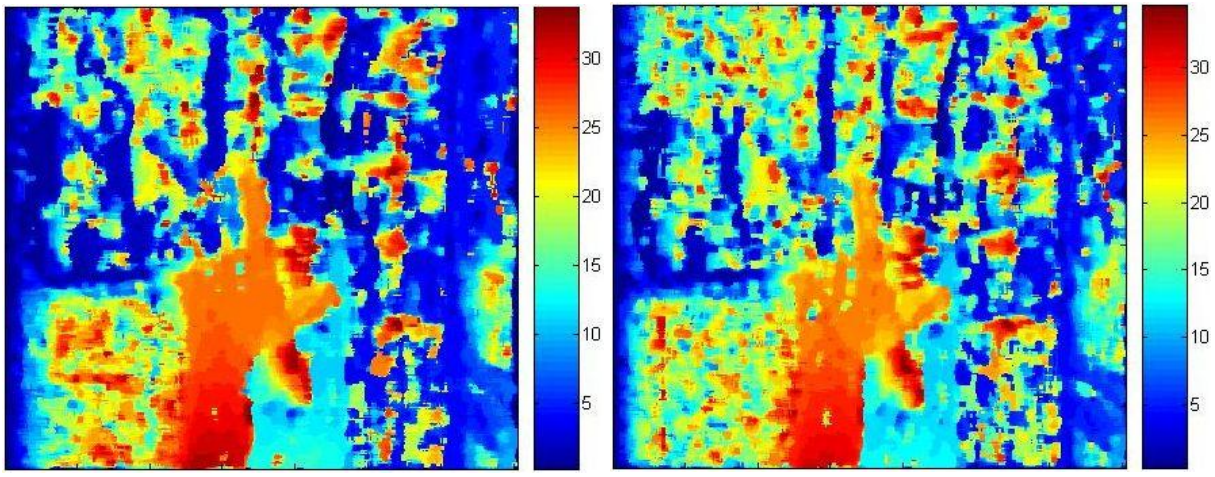
Figura 4.2.13. Mapas de disparidad al aplicar un filtro de mediana de tamaño *7x7* tras emplear el algoritmo *Census Disperso* sobre la ventana de coste para el par 1 (a) y el par 2 (b).

Se puede observar que, tras aplicar un filtro de mediana de tamaño *7x7*, se reduce en gran medida el ruido presente en el mapa original resultante de aplicar el algoritmo *Census Disperso* empleando un patrón de 2 píxeles con ventanas de trabajo de tamaño *7x7* sobre ambos pares de imágenes reales, con lo cual se pueden confirmar las conclusiones alcanzadas en el capítulo anterior para las imágenes de *Middlebury*, siendo preferible emplear el algoritmo *Census Disperso* aplicando el patrón de dispersión sobre la ventana de coste, y utilizando un patrón con el menor número de píxeles posible, reduciendo al mínimo de esta manera el tiempo de ejecución requerido para el procesamiento, y aplicando posteriormente un filtro de mediana para mejorar la calidad del mapa resultante.

Con la ayuda de la función *imagesc* de *Matlab* se puede apreciar de forma visual con mayor facilidad la comparativa entre los resultados obtenidos mediante la aplicación de los algoritmos *Census*, *mini-Census* y *Census Disperso*, la cual se ilustra a continuación en las *figuras 4.2.14* y *4.2.15* para ambos pares de imágenes, respectivamente. Cabe mencionar que en los casos de *mini-Census* y de *Census Disperso* los mapas mostrados son los obtenidos tras aplicar el filtro de mediana de tamaño *7x7* en la etapa de post-procesado, y particularmente el caso de *Census Disperso* es el correspondiente a la aplicación del patrón de dispersión sobre la ventana de coste.

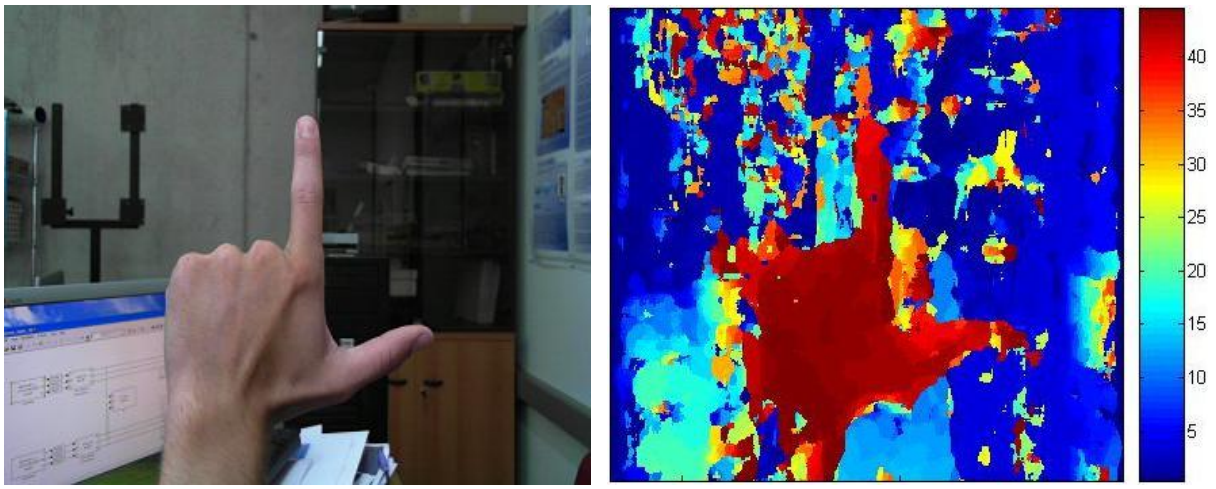


(a) (b)

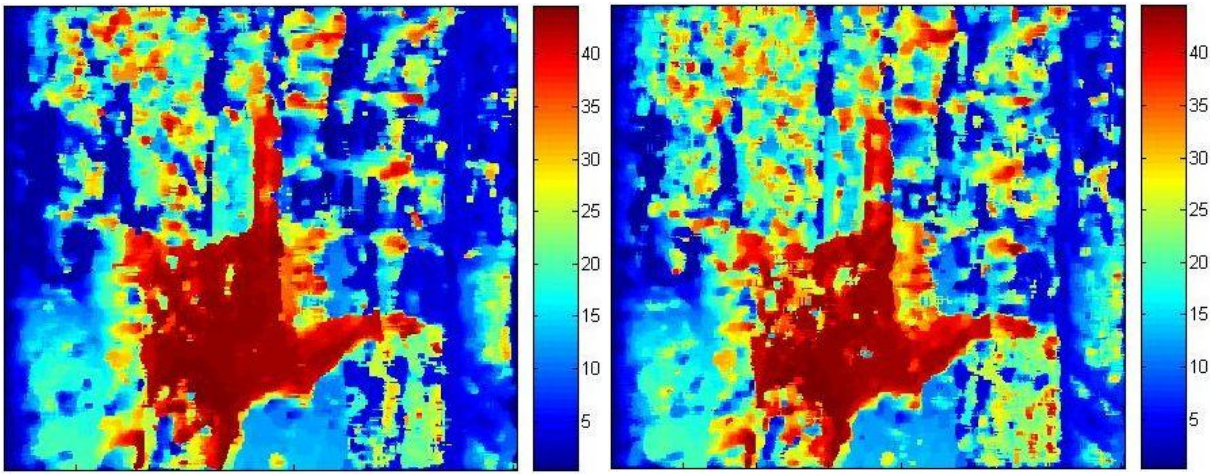


(c) (d)

Figura 4.2.14. Mapas de disparidad al aplicar *Census* (b), *mini-Census* (c) y *Census Disperso* (d) sobre el par 1.



(a) (b)



(c) (d)

Figura 4.2.15. Mapas de disparidad al aplicar *Census* (b), *mini-Census* (c) y *Census Disperso* (d) sobre el par 2.

Observando los resultados se puede afirmar que el mapa de mejor calidad es el obtenido por el algoritmo *Census* en su versión original. No obstante, es importante notar que los resultados proporcionados por los algoritmos *mini-Census* y *Census Disperso* son bastante similares entre sí, y están muy cercanos a la calidad del mapa proporcionado por *Census*, con lo cual se confirman las conclusiones alcanzadas en el capítulo anterior para las imágenes de *Middlebury*, siendo factible emplear los algoritmos *mini-Census* o *Census Disperso* aplicando una etapa de post-procesado con un filtro de mediana, aprovechando de esta forma la gran ventaja que proporcionan sendos algoritmos en lo que a tiempo de ejecución se refiere y permitiendo así obtener un resultado muy cercano al proporcionado por *Census* en su versión original en el menor tiempo posible.

Sería interesante contrastar el mapa de disparidad conseguido por el algoritmo *Census* con el proporcionado por las funciones propias de *Matlab* para ambos pares de imágenes reales, lo cual se muestra a continuación en la *figura 4.2.16*:

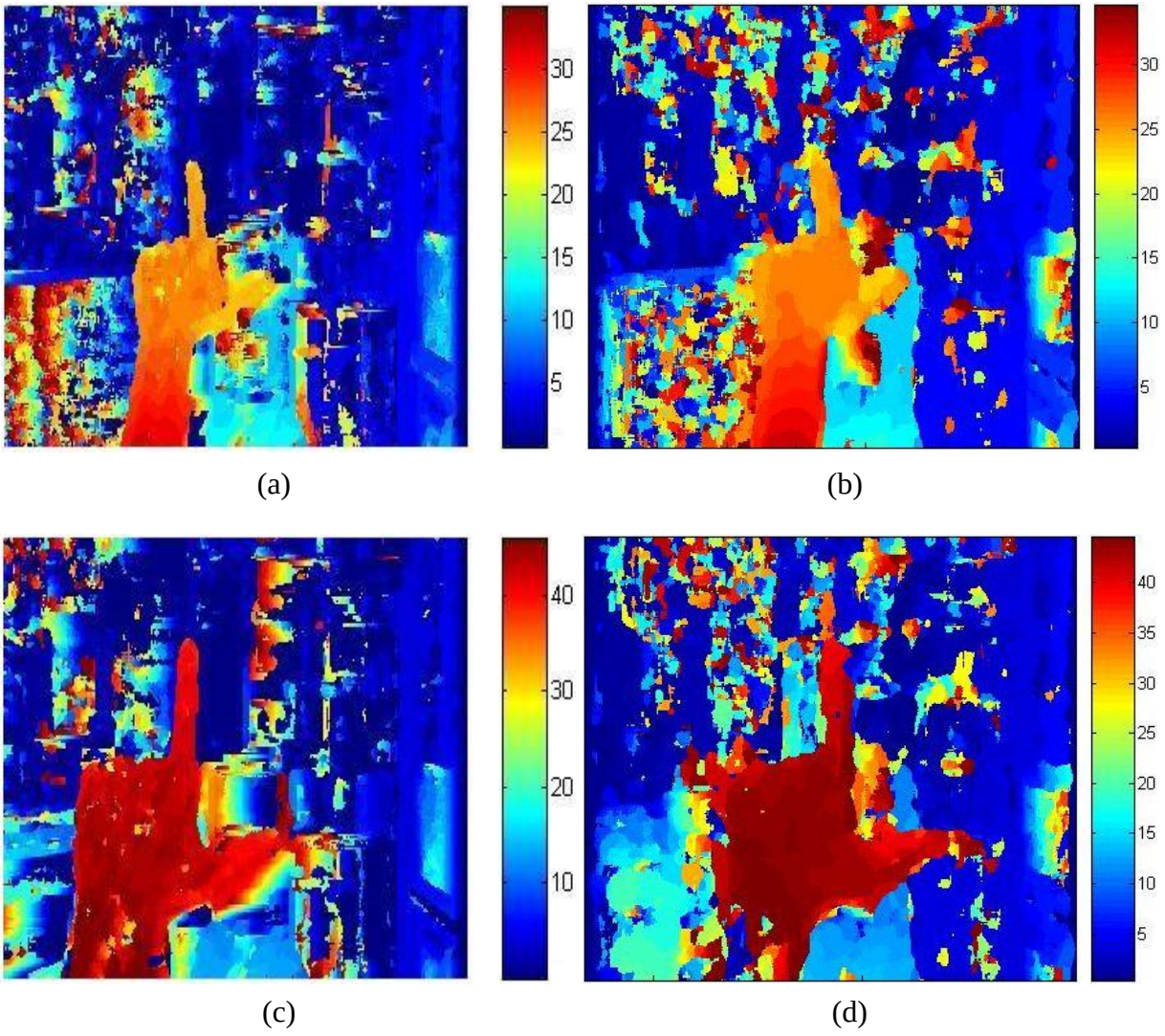


Figura 4.2.16. Mapas de disparidad obtenidos por *Matlab* para el par 1 (a) y para el par 2 (c), y por *Census* original para el par 1 (b) y el par 2 (d).

Aunque se puede observar que el resultado proporcionado por *Matlab* es ciertamente mejor, se trata no obstante de un buen resultado, ya que de nuevo se puede distinguir con cierta claridad el área delimitada por la mano como objeto principal en las escenas.

Para finalizar el presente capítulo, se muestra a continuación en la *figura 4.2.17* una comparativa para ambos pares de imágenes reales entre los resultados proporcionados por el algoritmo *SAD* básico y los obtenidos mediante el algoritmo *Census* en su versión original:

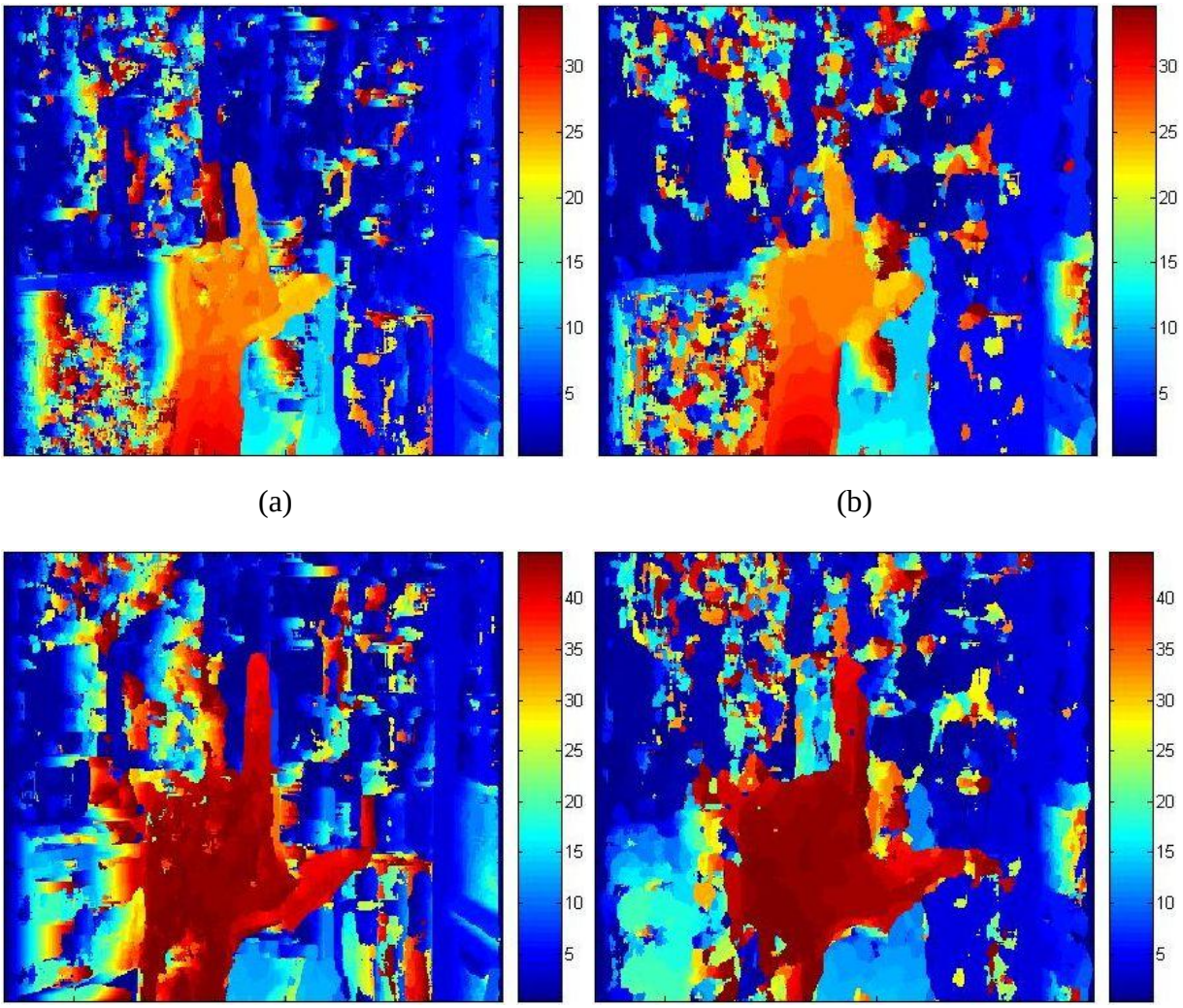


Figura 4.2.17. Mapas de disparidad obtenidos por *SAD* para el par 1 (a) y para el par 2 (c), y por *Census* original para el par 1 (b) y el par 2 (d).

A la vista de los resultados obtenidos, se pueden confirmar las conclusiones alcanzadas en el capítulo anterior para las imágenes de *Middlebury*, afirmando que el algoritmo *Census* no proporciona mejores resultados que *SAD*, por lo cual, es preferible hacer uso de los algoritmos basados en el criterio *SAD* en lugar de los basados en el criterio *Hamming*, puesto que proporcionan mapas de disparidad de mejor calidad y su coste computacional es mucho menor.

CAPÍTULO 5. CONCLUSIONES Y LÍNEAS FUTURAS

A lo largo de los capítulos del presente informe, se ha llevado a cabo un análisis exhaustivo de una gran diversidad de algoritmos basados en área para la obtención de mapas de disparidad a partir de imágenes estereoscópicas, agrupados según sus respectivos criterios de coste, ya sea aquellos basados en el criterio *SAD* o los basados en el criterio *Hamming*. Tras la conclusión de este meticuloso análisis, observando los resultados obtenidos mediante las pruebas realizadas con dichos algoritmos tanto sobre las imágenes prediseñadas pertenecientes a la *Universidad de Middlebury* como sobre las imágenes reales y específicas tomadas en los laboratorios de la *Universidad Politécnica de Cartagena* mediante el sistema *webcam* estéreo constituido para tal propósito, se han podido alcanzar una serie de conclusiones, tanto de carácter general, como a nivel particular dentro de cada uno de los grupos de algoritmos.

De forma general, se puede afirmar que el empleo de esta técnica de visión estereoscópica es perfectamente factible a la hora de su aplicación para mejorar la robustez del algoritmo de reconocimiento de la mano, pues las tasas de acierto de los algoritmos empleados según las características y parámetros utilizados para su ejecución en el caso de las imágenes prediseñadas con respecto a los mapas de disparidad ideales oscilan entre el 80% y el 95% lo cual indica una gran eficacia, mientras que por otro lado al aplicarlos sobre las imágenes reales se puede distinguir visualmente con cierta claridad el área constituida por la mano como objeto principal en las escenas por encima del resto de objetos situados más al fondo. Así mismo, tras la comparativa realizada entre los algoritmos basados en el criterio *SAD* y los basados en el criterio *Hamming* se puede concluir en que es preferible emplear los pertenecientes al primer grupo (*SAD* básico, *Rank* y *Soft-Rank*) puesto que la eficacia demostrada por los mismos ha resultado ser mayor que la observada para los algoritmos basados en *Hamming*, con la ventaja añadida de que su coste computacional es mucho menor gracias a su mayor simplicidad.

A nivel particular, para el caso de los algoritmos basados en el criterio de coste *SAD*, se puede concluir en que las versiones de *SAD* que hacen uso de las transformadas *Rank* y *Soft-Rank* no proporcionan resultados notablemente mejores que la versión básica del algoritmo, por lo cual es preferible emplear dicha versión, o en todo caso la versión con la transformada *Rank*, aplicando la técnica de reutilización de valores para reducir en la medida de lo posible el tiempo de ejecución demandado por el algoritmo.

En lo que respecta a los algoritmos basados en el criterio de coste *Hamming*, se concluye que lo más apropiado es emplear el algoritmo *mini-Census* aplicando posteriormente una etapa de post-procesado con un filtro de mediana de tamaño no demasiado amplio (*5x5* o *7x7*), ya que se ha demostrado que la aplicación del filtro de mediana produce una mejora en la tasa de acierto de los mapas resultantes de al menos

entre un 5% y un 10%, con lo cual se puede llegar a superar la calidad de los resultados obtenidos mediante el algoritmo *Census* en su versión original, con la ventaja añadida de reducir considerablemente el tiempo de ejecución requerido para el procesamiento, puesto que la versión *mini-Census* requiere mucha menos carga computacional. Así mismo, también es una gran opción emplear el algoritmo *Census Disperso*, aplicando un patrón de dispersión de 2 o 4 píxeles sobre la ventana de coste, reduciendo de esta manera al mínimo el tiempo de ejecución requerido para el procesamiento y consiguiendo una calidad en el mapa de disparidad bastante cercana a la lograda mediante el algoritmo *Census* en su versión original, aplicando también un filtro de mediana como etapa de post-procesado. Como tercera y última opción, se puede emplear el algoritmo *Census* original aplicando la técnica de reutilización de valores para reducir en la medida de lo posible el tiempo de ejecución demandado por el algoritmo.

Cabe también destacar que, a pesar de que la eficiencia de los algoritmos no pretendía ser objeto de este proyecto, se ha procedido también a su análisis para poder así proporcionar unas conclusiones más robustas a la hora de afirmar la conveniencia de emplear uno u otro, y se han propuesto, implementado y analizado versiones de los algoritmos en ambos grupos (*SAD* y *Hamming*) que permitieran producir un cierto impacto en la eficiencia de los mismos, tales como el uso de la técnica de reutilización de valores o el algoritmo *Census Disperso*, los cuales han resultado ser exitosos, habiendo conseguido reducir el tiempo de ejecución hasta 10 órdenes de magnitud en algunos casos.

En cuanto a líneas futuras, el presente proyecto abre un amplio abanico de posibilidades para futuros trabajos, puesto que la información aportada por este tipo de técnicas de visión estereoscópica es de gran interés en multitud de aplicaciones y sistemas de visión artificial, para medición de distancias a objetos o reconstrucción de escenas 3D. Sería interesante realizar más pruebas con respecto a los algoritmos aquí presentados, como por ejemplo evaluar los resultados del algoritmo *Census Disperso* cuando se realiza una combinación de las 2 variantes del mismo implementadas para este proyecto, aplicando en primer lugar un patrón sobre la ventana de transformación y en segundo lugar otro patrón (igual o distinto) sobre la ventana de coste. También sería interesante estudiar otros tipos de filtros distintos al filtro de mediana para aplicarlos como etapa de post-procesado sobre los mapas obtenidos por los algoritmos. Además, convendría ampliar la base de datos de imágenes reales y específicas analizando los resultados producidos por los algoritmos al variar determinados parámetros como por ejemplo la distancia de separación entre ambas cámaras del sistema estereoscópico.

También como línea futura se plantea la optimización del tiempo de ejecución de los algoritmos, que, a pesar de los buenos resultados proporcionados por las versiones implementadas en el presente proyecto para reducir el tiempo de procesamiento, hay que decir que los tiempos logrados siguen sin ser apropiados para una aplicación que pretenda funcionar en tiempo real, como bien puede ser el algoritmo

de reconocimiento de la mano. Por lo tanto, una vez comprobada la eficacia de los algoritmos, el siguiente paso sería trabajar en la optimización de su ejecución, lo cual, con la ayuda de las técnicas y versiones de los algoritmos aquí implementadas (como la reutilización de valores o el algoritmo de *Census Disperso*) es una tarea que seguramente a día de hoy puede ser relativamente sencilla de emprender y concluir con resultados satisfactorios para el propósito perseguido, pudiendo implementar los algoritmos de visión estereoscópica en aplicaciones que se ejecuten en tiempo real, gracias a las librerías existentes para otras plataformas de programación (como *OpenCv* para el lenguaje *C*) que contienen funciones específicamente diseñadas y optimizadas para la aplicación de determinadas técnicas de visión artificial y tratamiento de imagen e incluso el empleo de técnicas de *GPU Computing* que hoy por hoy se encuentran bastante desarrolladas y su uso en visión por computador crece a un ritmo vertiginoso de tal forma que, a juicio personal del autor de este proyecto, está empezando a no entenderse la visión artificial sin el uso de la *GPU*.

En última instancia, cabría proponer además como línea futura la adaptación del algoritmo original de reconocimiento de la mano para que haga uso de la técnica de visión estereoscópica aquí estudiada, ya sea realizando la sustitución de la segmentación basada en el color de la piel que actualmente emplea el algoritmo por ésta, o bien implementando una combinación de ambas técnicas, con el objeto de analizar los resultados obtenidos y concluir definitivamente en la validez del procesado.

BIBLIOGRAFÍA

[1] J. Toledo, J. Martínez, J. Garrigós y J. Ferrández, «Implementación de un sistema de reconocimiento de la mano» *Jornadas de Computación Reconfigurable y Aplicaciones,* pp. 223-230, 2010.

[2] J. Toledo, J. MArtínez, J. Garrigós, J. Ferrández y V. Rodellar, «Skin color detection for real time mobile applications» *IEEE Conf. on Field Programmable Logic and Applications,* pp. 712-724, 2006.

[3] J. Toledo, J. Martínez y J. Ferrández, «Hand-based Interface for Augmented Reality» *IEEE Symposium on Field-Programmable Custom Computing Machines,* pp. 291-292, 2007.

[4] «Middlebury Stereo Vision Page» http://vision.middlebury.edu/stereo/.

[5] R. Szeliski, «Computer Vision: Algorithms and Applications», pp. 534-575, 2010.

[6] D. Scharstein y R. Szeliski, «High-accuracy stereo depth maps using structured light» de *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

[7] D. Scharstein y R. Szeliski, «A taxonomy and evaluation of dense two-frame stereo correspondence algorithms» *International Journal of Computer Vision,* pp. 7-42, 2002.

[8] K. Ambrosch y W. Kubinger, «Accurate hardware-based stereo vision» *Computer vision and image understanding,* 2010.

[9] M. Humenberger, C. Zinner, M. Weber, W. Kubinger y M. Vincze, «A fast stereo matching algorithm suitable for embedded real time systems» *Computer vision and image understanding,* 2010.

[10] W. Fife y J. Archibald, «Improved Census Transforms for resource-optimized stereo vision» *IEEE Transactions on Circuits and Systems for Video Technology,* 2013.

[11] M. Z. Brown, D. Burschka y G. D. Hager, «Advances in Computacional Stereo» *IEEE Transactions On Pattern Analysis And MAchine Intelligence,* 2003.

[12] «Matlab webpage. The Matworks» http://es.mathworks.com/.

[13] C. Colodro Conde, «Diseño e implementación hardware de algoritmos de estimación de la profundidad en visión estereoscópica», 2011.

[14] H. Hirchsmüler y D. Scharstein, «Evaluation of Stereo Matching Costs on Images with Radiometric Differences» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2009.

[15] J. M. Guerrero Hernández, G. Pajares Martinsanz y M. Guijarro Mata-García, «Técnicas de Procesamiento de Imágenes Estereoscópicas».

[16] M. Jo Hanna, «Computer Matching of Areas in Stereo Images», 1974.

[17] F. Lecumberry, «Cálculo de disparidad en imágenes estéreo, una comparación» *III Workshop de Computación Gráfica, Imágenes y Visualización*.

[18] L. di Stefano, M. Marchionni y S. Mattoccia, «A fast area-based stereo matching algorithm» *Image Vision Comput.,* pp. 983-1005, 2004.

[19] C. Colodro, J. Toledo, J. J. Martínez, J. Garrigós y J. M. Ferrández, «Evaluación de algoritmos de correspondencia estereoscópica y su implementación en FPGA».

[20] N. Yen-Chung Chang, T.-H. Tsai, B.-H. Hsu, Y.-C. Chen y T.-S. Chang, «Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight» *IEEE Transactions on Circuits and System for Video Technology,* 2010.

[21] R. Zabih y J. Woodfill, «Non-parametric Local Transforms for Computing Visual Correspondence» *In Proceedings of European Conference on Computer Vision*, 1994.